Xtreme9.0 - Block Art

An editorial for this problem is available at the bottom of this page.

The NeoCubist artistic movement has a very distinctive approach to art. It starts with a rectangle which is divided into a number of squares. Then multiple rounds of layering and scraping occur. In a layering round, a rectangular region of this canvas is selected and a layer of cubes, 1 cube deep, is added to the region. In a scraping round, a rectangular region of the canvas is selected, and a layer of cubes, again 1 cube deep, is removed.

The famous artist I.M. Blockhead seeks your help during the creation of this artwork. As he is creating his art, he is curious to know how many blocks are in different regions of the canvas.

Your task is to write a program that tracks the creation of a work of Pseudo-Cubist art, and answers I.M.'s periodic queries about the number of blocks that are on the canvas. Consider, for example, the following artwork created on a canvas with 5 rows and 15 columns or squares. The canvas starts without any blocks, like in the figure below. We label cells in the canvas based on the tuple (row,column), with the upper left corner being designated (1,1). The numbers in each cell represent the height of the blocks in that cell.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

After adding a layer in blocks to the rectangle with upper left corner at (2,3) and a lower right corner of (4, 10), the canvas now looks like the following:

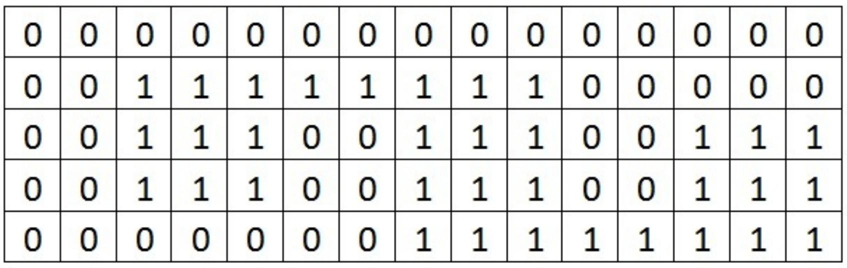
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

After adding a layer of blocks in the rectangle with upper left corner at (3,8) and a lower right corner of (5, 15), the canvas now looks like the following:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	1	1	1	1	1	2	2	2	1	1	1	1	1
0	0	1	1	1	1	1	2	2	2	1	1	1	1	1
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

If Blockhead were to ask how many blocks are currently in the artwork in the rectangle with upper left corner (1,1) and lower right corner (5,15), you would tell him 48.

Now, if we remove a layer of blocks from the rectangle with upper left corner at (3,6) and a lower right corner of (4, 12), the canvas now looks like the following:



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	1	1	1	0	0	1	1	1	0	0	1	1	1
0	0	1	1	1	0	0	1	1	1	0	0	1	1	1
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

If Blockhead were to ask how many blocks are now in the artwork in the rectangle with upper left corner (3,5) and lower right corner (4,13), you would tell him 10.

“Beautiful!” exclaims Blockhead.

Input Format

The first line in each test case are two integers *r* and *c*, $1 <= r <= 12$, $1 <= c <= 10^6$, where *r* is the number of rows and *c* is the number of columns in the canvas.

The next line of input contains an integer *n*, $1 <= n <= 10^4$.

The following *n* lines of input contain operations and queries done on the initially empty canvas. The operations will be in the following format:

[operation] [top left row] [top left column] [bottom right row] [bottom right column]

[operation] is a character, either “a” when a layer of blocks is being added, “r” when a layer of blocks is being removed, and “q” when Blockhead is asking you for the number of blocks in a region.

The remaining values on the line correspond to the top left and bottom right corners of the rectangle.

Note: You will never be asked to remove a block from a cell that has no blocks in it.

Output Format

For each “q” operation in the input, you should output, on a line by itself, the number of blocks in the region of interest.

Sample Input

```
5 15
5
a 2 3 4 10
a 3 8 5 15
q 1 1 5 15
r 3 6 4 12
q 3 5 4 13
```

Sample Output

```
48
10
```

Explanation

This test case corresponds to the description given above.

Editorial

The following editorial explains an approach for solving this problem.

An Inefficient Approach

One common approach to this problem was to attempt to store the canvas as a two dimensional array of integers, in which each cell in the array would be equal to the number of blocks in each cell on the canvas. This approach mirrors the figures provided in the *Problem Statement*.

The problem with this approach is that the canvas can be as large as 12 by 10^6. In addition, there can be up to 10^4 operations that may cover all of this canvas. The number of times a cell is accessed or changed, therefore, could be as high as $12 * 10^6 * 10^4 = 1.2 * 10^{11}$, or 120 billion. This approach would certainly result in a time limit exceeded on the largest test cases.

An Efficient Approach

Rather than storing the canvas, a better approach is to store two collections of rectangles, an `added` collection representing rectangles of blocks added to canvas, and a `removed` collection representing rectangles of blocks removed from the canvas.

Whenever you encounter a query in input, you create a `sum` variable, and set it equal to 0. You then iterate through the `added` collection, and increase the value in `sum` by the size of the intersection between the query rectangle and each rectangle in `added`. Then, you would iterate through the `removed` collection, and reduce the value in `sum` by the area of intersection between the query rectangle and each rectangle in `removed`. The result of the query is the value in `sum`.

Note that in the worst case, the program would create 5000 rectangles in the `added` and `removed` collections, and then perform 5000 queries on these rectangles, resulting in $2.5 * 10^7$ comparisons between rectangles. This small number of comparisons can easily be completed within the time limits provided.