Summer camp

Every year Ivancho takes part in a summer camp and this time isn’t an exception. Traditionally, the organisers of the camp conduct a “game of regions”. The kids are divided by teams as every day one of the teams is playing the game. The game is played on **NxN** grid consisted of square cells with size **1**. In the very beginning of each daily round, the grid is almost empty. In some of the cells there are flags with a written single positive number **C** on each flag.

The aim of the game is to divide the grid into regions following some rules:

1. Each region must contain at least one flag and must be connected. In other words, from each cell of the region you could reach each other cell moving only trough cells of the region. You can move from one cell to another if and only if they have common side.
2. It is possible that one region contains more than one flag. The only constraint is that all flags in a region should have same numbers written on them.
3. Each region must contain exactly the same amount of cells as the number written on the flags placed in it.

1. In the end there could be flags which aren’t included in any region.
2. In the end there could be cells which aren’t included in any region.

When all teams took their turns to play the game, the team which succeed in dividing the grid on regions with maximal sum of the areas wins.

Today, Ivanchos team is playing the game and as you may already guessed he wants to win pretty much. That’s why Ivancho is asking you for help. Write a program **regions**, which gets the initial state of the grid and divides it into regions following the rules above in a way that the amount of cells in all regions is as big as possible.

**Input:** The first row of the input file **regions.in** contains two integers **N** and **R** where **N** is the length of the grid and **R** is the number of given flags, each of them containing a number. The following **R** rows contain three integers - **Xi**, **Yi** and **Ci, (i = 1, 2, …, R)** – respectively the coordinates of the flags (X**i** - row index; Y**i** - column index) and the number written on the flag. **Remark:** The input is 1-indexed.

**Output:** The output file **regions.out** should contain a NxN matrix which shows the division by regions. Every cell of the matrix contains one positive integer – the id of the region to which it belongs to or 0 if the cell doesn’t belong to any region (doesn’t matter if the cell contains a flag or not). Each region needs to have its own id, where if we have K regions, the ids are integers from 1 to K. Otherwise the output is invalid. It’s not allowed to have more than one region with the same id. Otherwise the output will be invalid as well.

**Limits:** 1 ≤ X ≤ N, 1 ≤ Y ≤ N, 1 ≤ C ≤ N2.

**Time limit:** 5 sec

**Memory limit:** 256 MiB

**Test groups:**

|  |  |  |  |
| --- | --- | --- | --- |
| Group | % tests | N | R |
| 1 | 25 | 5 ≤ N ≤ 25 | N ≤ R ≤ 2N |
| 2 | 25 | 25 ≤ N ≤ 50 | 2N ≤ R ≤ 4N |
| 3 | 25 | 50 ≤ N ≤ 75 | 4N ≤ R ≤ 8N |
| 4 | 25 | 75 ≤ N ≤ 100 | 8N ≤ R ≤ 16N |

**Grading:**

If the output on a given test is invalid, your solution will receive 0 score for the test. Otherwise the score is calculated by the following formula:

 score = number of squares with nonzero id.

If your score is *yourScore* and the maximum score received by any other contestant is *maxScore* then your solution will receive (yourScore + 1) / (maxScore + 1) percentage of the points for the corresponding test.

**Example:**



|  |  |
| --- | --- |
| **regions.in** | **regions.out** |
| 5 141 1 31 4 61 5 62 1 52 3 42 5 63 2 54 2 13 3 43 5 64 4 65 3 35 4 15 5 2 | 1 1 1 2 23 3 6 6 23 3 6 2 23 4 6 2 57 7 7 8 5 |

**Note:** The matrix in regions.out differs from the solution on the picture. The matrix contains ids which are unique for each region. We have 8 regions so to meet the required rules the ids are values from 1 to 8. The solution on the picture represents each region by its size instead of id.