A . Grid Compression

Limits: 4 sec., 1024 MiB

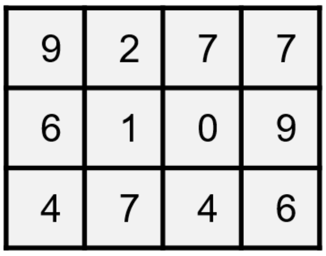
In traditional positioning based fingerprint, the amount of memory occupied has always been the focus of attention and it is strongly related to the actual number of grids. For the same area, the smaller the number of grids, the smaller the memory space occupied. Therefore, under certain conditions, it is necessary to compress the grid as much as possible.

You are given a grid matrix with **H** rows and **W** columns. Rows are numbered top-to-bottom from 0 to $\mathbf{H}-1$ and columns are numbered left-to-right from 0 to $\mathbf{W}-1$. A grid at row r and column c has a corresponding number of samples $\mathbf{S}_{r,c}$. For example, check the sample matrix in figure 1a.

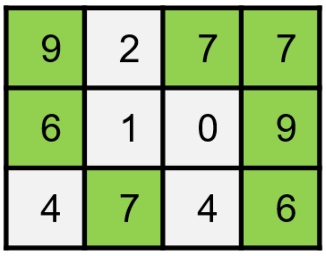
You are also given integer **T** which is the minimal required number of samples in a grid. If $S_{r,c} \geq \mathbf{T}$ then the grid at (r, c) is valid. For $\mathbf{T} = 5$, the figure 1b shows all valid grids in the sample matrix.

Figure 1: Grids

(a) Sample matrix



(b) Valid grids



Your task is to compress some of the grids into valid rectangles. Each rectangle size must be $\mathbf{M} \times \mathbf{N}$ or $\mathbf{N} \times \mathbf{M}$. Rectangles cannot overlap but they can cover some grids outside of the initial matrix (assume all such grids have number of samples equal to 0, see figure 2a). Besides, they don't have to fully cover the entire matrix. A rectangle is valid if its average number of samples per grid is at least **T**, more formally rectangle α is valid if and only if

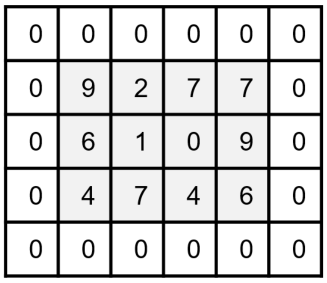
$$\sum_{(r,c)\in\alpha} \mathbf{S}_{r,c} \geq \mathbf{T} \cdot \mathbf{N} \cdot \mathbf{M}.$$

For example, if $\mathbf{T} = 5$, $\mathbf{N} = 1$ and $\mathbf{M} = 3$ then two valid rectangles are illustrated in figure 2b.

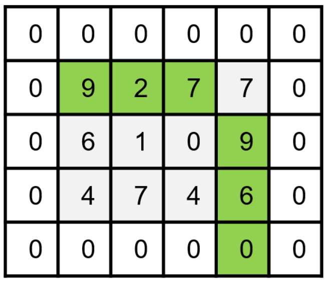
Your goal is to place as many valid rectangles as possible.

Figure 2: Rectangles

(a) Extended grid matrix



(b) Valid rectangles



Input

The first line contains two integers **H** and **W**. In the second line there are two integers **N** and **M**. And the third line contains integer **T**.

The next **H** lines describes the grid matrix. The r-th of them (0-based) contains **W** integers $\mathbf{S}_{r,0}, \mathbf{S}_{r,1}, \ldots, \mathbf{S}_{r,\mathbf{W}-1}$ separated by single spaces. Here $\mathbf{S}_{r,c}$ is the number of grid samples at row r and column c.

Output

In the first line print the number of rectangles $\mathbf{X}$. In each of the following $\mathbf{X}$ lines print four intgers $\mathbf{r_1}$, $\mathbf{c_1}$, $\mathbf{r_2}$ and $\mathbf{c_2}$ separeted by single spaces which describe a single valid rectangle. Here the rectangle top left grid is located at row $\mathbf{r_1}$ and column $\mathbf{c_1}$ while its bottom right grid is located at row $\mathbf{r_2}$ and column $\mathbf{c_2}$.

Constraints

$1 \leq \mathbf{H}, \mathbf{W} \leq 250$,
$1 \leq \mathbf{N}, \mathbf{M} \leq 10$,
$1 \leq \mathbf{T} \leq 100$,
$0 \leq \mathbf{S}_{r,c} \leq 100$,
for each rectangle $\mathbf{r_1} \leq \mathbf{r_2}$ and $\mathbf{c_1} \leq \mathbf{c_2}$,
each rectangle ($\mathbf{r_1}$, $\mathbf{c_1}$, $\mathbf{r_2}$, $\mathbf{c_2}$) size must be $\mathbf{M} \times \mathbf{N}$ or $\mathbf{N} \times \mathbf{M}$,
each rectangle ($\mathbf{r_1}$, $\mathbf{c_1}$, $\mathbf{r_2}$, $\mathbf{c_2}$) must be valid,
all rectangles ($\mathbf{r_1}$, $\mathbf{c_1}$, $\mathbf{r_2}$, $\mathbf{c_2}$) must not overlap.

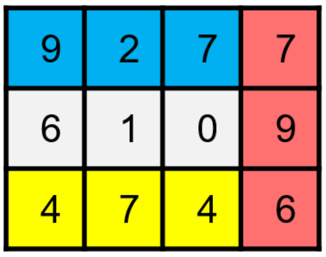
Samples

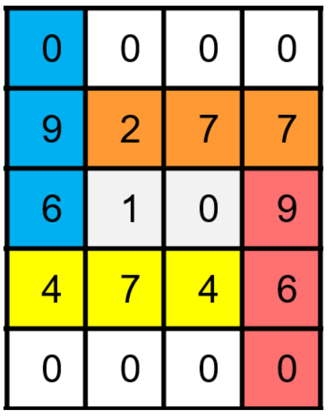
Input (*stdin*)	Output (*stdout*)

3 4
1 3
5
9 2 7 7
6 1 0 9
4 7 4 6
	3
	0 0 0 2
	2 0 2 2
	0 3 2 3

Notes

Figure 3: Sample case


(a) Sample output


(b) One of the optimal answers

You can print any valid answer your solution can find, not necessarily the most optimal one. For example, check the sample case input matrix which is shown in figure 1a. The provided output illustrated in figure 3a is valid but not optimal. It is possible to place 4 valid rectangles as shown in figure 3b. Then the output would be

```
4
-1 0 1 0
0 1 0 3
2 0 2 2
1 3 3 3
```

Scoring

Your raw score for a test case is the total number of valid rectangles **X** in your output. If your output was invalid (no output, time limit exceeded, run time crash, wrong output format, wrong number of rectangles, wrong rectangle size, overlapping rectangles etc.) then your raw score on this test case will be -1.

If your raw score for a test case is less than or equal to 0 then your normalized score for that test case is 0. Otherwise, your normalized score for each test case is

$$\left\lfloor \frac{RAW \cdot 10^7}{MAX + 1} \right\rfloor,$$

where RAW is your raw score and MAX is

$$\left\lfloor \frac{\sum_{r=1}^{\mathbf{H}} \sum_{c=1}^{\mathbf{W}} \mathbf{S}_{r,c}}{\mathbf{T} \cdot \mathbf{N} \cdot \mathbf{M}} \right\rfloor.$$

Note that RAW is always less than or equal to MAX which means your normalized score for a test case is less then 10^7.

Finally, your overall score is the sum of your normalized score over all test cases.

Submissions

- The execution time limit is 4 seconds per test case and the memory limit is 1024 mibibytes.
- The code size limit is 64 kibibytes.
- The compilation time limit is 1 minute.
- There are 100 provisional test cases. Your submissions will be evaluated on provisional set during the submission phase.
- You can submit your code once per 10 minutes and you will get a feedback with your normalized score for each of the provisional tests.
- There will be 1000 test cases in the final testing after the submission phase is over. The final results will be announced during one week.

Quick start

Check the sample solution which simply looks for the first **N** × **M** valid rectangle and prints it. The source code is available for all contest programming languages:

- C++
- Java
- C#
- Pascal
- Python
- F#

Test generator

All test cases including provisional and final test sets are generated using the test generator. Each test case is produced by providing the generator with a seed number. Seed number 1 corresponds to the sample case. You do NOT know seed numbers for provisional or final test sets, but you can use the generator for a local testing.

1. Check the generator source code
2. Save it to file named Generator.java
3. Compile the source code with Java

   ```
   javac Generator.java
   ```

4. Generate a test case for some seed number

   ```
   java Generator <seed>
   ```

   This will print the test case for <seed> as well as write a visualization image to file named grid-<seed>.png in your working directory.