A. Multi-core message processing system

Limits: 4 sec., 1024 MiB

As a multi-core message processing system, a 5G base station must not only pursue message processing capabilities, but also processing efficiency. That is, the higher the *affinity* and *capability*, the better.

Your task is to design a scheduler to allocate N input messages to M processing cores. When a core is allocated a message, its processing cannot be preempted, and the next message cannot be processed until the current one is done. The processing capacity of each core is the same.

Each message carries the following fields:

- $MsgType$: message type processed by the base station, integer number in range [1, 200]. The *affinity* increases, if a core consequtively processes messages of the same type.
- $UsrInst$: user instance number accessed by the base station, integer number in range [1, 10000].
- $ExeTime$: time consumed by a core to process this message, simplified to an integer number in range [1, 2000].
- $DeadLine$: an integer number in range [1, 10^9]. To increase the *capability*, the message has to be processed not later than its deadline.

The system guarantees that each pair ($MsgType$, $UsrInst$) is unique.

There are two requirements you must follow:

1. All messages of the same user instance must be allocated to the same core.
2. The message relative order of a user instance must be preserved, i.e. it should be the same as in the input.

Input

The first line contains three integers N, M, and C. Here C is the processing time, i.e. the global deadline. Each of the next N lines describes a message and contains four integers: $MsgType$, $UsrInst$, $ExeTime$, and $DeadLine$. Numbers in a line are separated with single spaces.

Output

Print M lines corresponding to each of the cores. In each line, first print the number of messages allocated to this core. Then for each message print two numbers: $MsgType$ and $UsrInst$. Separate numbers in a line with single spaces.

Constraints

$1 \le N \le 100000$ (10^5),
$1 \le M \le 30$,
$1 \le C \le 2147483647$ ($2^{31} - 1$),
$1 \le MsgType \le 200$,
$1 \le UsrInst \le 10000$,
$1 \le ExeTime \le 2000$,
$1 \le DeadLine \le 10^9$.

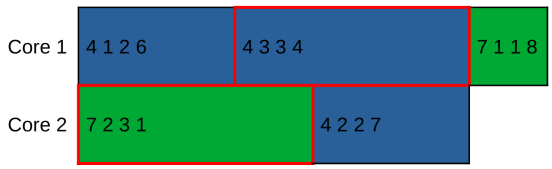
Samples

Input (*stdin*)	Output (*stdout*)
5 2 9 4 1 2 6 7 2 3 1 4 3 3 4 7 1 1 8 4 2 2 7	3 4 1 4 3 7 1 2 7 2 4 2

Notes

You can print any valid answer your solution can find, not necessarily the most optimal one.

The figure below illustrates the sample output. Here, the time goes from left to right, each rectangle represents a message processed at a core, different colors correspond to different message types, and messages processed after their deadline have red borders.



For the sample test case, here is an example of allocation which breaks the the first requirement, as messages of user instance 1 are allocated to different cores.

```
2 4 1 4 3
3 7 2 4 2 7 1
```

The next example does not satisfy the second requirement, since the relative message order of user instance 2 is reversed.

```
3 4 1 4 3 7 1
2 4 2 7 2
```

Scoring

If the allocation you printed is invalid or does not satisfy the requirements, your score for the test is 0 points. Otherwise let $AffinityScore$ be the total number of messages preceeded by a message (in the corresponding core message sequence) of the same type in your allocation. Also let $CapabilityScore$ be the total number of messages processed without beyond the deadline, i.e. minimum of the message deadline and the global deadline. Your score for the test is calculated as

$$\left\lfloor \frac{AffinityScore + CapabilityScore}{2 \cdot N} \cdot 10^7 \right\rfloor.$$

Your overall score is the sum of your scores over all test cases.

For the sample output, the $AffinityScore$ is 1, $CapabilityScore$ is 3, and the test score is

$$\left\lfloor \frac{1+3}{2 \cdot 5} \cdot 10^7 \right\rfloor = 4000000.$$

Note that it is possible to increase *capability*, if on core 1 we start with processing the message of user instance 3. The following output for the sample test case scores 5000000.

```
3 4 3 4 1 7 1
2 7 2 4 2
```

Submissions

- The execution time limit is 4 seconds per test case and the memory limit is 1024 mebibytes.
- The code size limit is 64 kibibytes.
- The compilation time limit is 1 minute.
- There are 50 provisional test cases. Your submissions will be evaluated on the provisional set during the submission phase.
- You can submit your code once per 10 minutes and you will get feedback with your score for each of the provisional tests.
- There will be 500 test cases in the final testing after the submission phase is over. The final results will be announced in one week.

Quick start

Check the sample solution which iterates messages from the input and if a core is not yet assigned for the current user instance, it will choose the one with the minimum total processing time. The source code is available for some of the contest programming languages:

- C++
- Python
- Java
- C#

Tests

All test cases including provisional and final sets are generated by the generator. A 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 local testing by taking a few simple steps:

1. Check the generator source code
2. Save it to a 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>` to *standard output*.