Servers (Challenge)

Problem Code: **SRVRS**

Read problems statements in Mandarin Chinese (http://www.codechef.com/download/translated/JULY17/mandarin/SRVRS.pdf), Russian (http://www.codechef.com/download/translated/JULY17/russian/SRVRS.pdf) and Vietnamese (http://www.codechef.com/download/translated/JULY17/vietnamese/SRVRS.pdf) as well.

The Chef and Batman are working on the new computer network for the Batcave. The Batcave can be thought as a 2D plane and it contains all **N** servers (advanced host machines). Every server is located at a single point on this 2D plane and can't change places (the computers are very heavy so moving them is next to impossible). Each server has a few CPUs (central processing unit), each having some processing time (i.e. the time it takes to run a task). You can consider that all tasks require the same resources (two different tasks that run on the same CPU of the same server take the same time to execute). After many hours of hard work, the network seems ready and our two heroes want to test it using a slightly complicated test framework.

Their test consists of **Q** queries. Each query is actually a task (given as a point in the same 2D plane). This task has to be executed on some available CPU (of some server in the network). A CPU (of some server) is available if there is **no** task that currently runs on it. For example, if a task arrives at time **5** and is executed on a CPU with a processing time of **10** then the CPU is busy in **[5..14]** (i.e. it is again available at time **15**). Any attempt to execute a task on a busy CPU will be punished with a network crash (and neither Chef nor Batman wants to reconfigure the whole network again).

There is a cost associated with scheduling (and running) a task. Let's assume that we have a server, located at **(a, b)**, with a CPU of processing time **p**. If you execute a task located at **(x, y)**, then the cost will be ***euclid_distance((a, b), (x, y)) + p*** (which is the cost of sending the task to the server plus the time to actually run the task on the CPU).

The test framework is not perfect and doesn't allow multiple tasks waiting to be executed at the same time (i.e. the network will receive another task only after all previous tasks have been scheduled and are running or have finished). We can say that the interaction between the testing framework and the network is **interactive**.

Assigning a task to a (server, cpu) pair is a hard task so Batman and Chef want you to help them.

Input

The first line of the input file contains two integers, **N** and **Q**, where **N** is the number of servers and **Q** is the number of queries.

The i-th of the next **N** lines contains the description of the i-th server. The servers are 1-indexed.

A server is described by a sequence of integers: **x y k p_0 p_1 ... p_{k-1}**. **x** and **y** represent the location of the server in the plane. **k** represents the number of CPUs the server has. The j-th of the next **k** integers indicates the processing time of its j-th CPU. The CPUs are 1-indexed.

The **Q** tasks can be obtained by using the command specified below, in the **Interactive** segment.

Output

The output file should contain **Q** lines, each line containing two integers: $\mathbf{s_i}$ and $\mathbf{c_i}$, where $\mathbf{s_i}$ is the index of the server and $\mathbf{c_i}$ is the index of the CPU on that server that runs the *i*th task.

Interactive

The problem is **interactive**. You will get the coordinates of each task one at a time after you have given an answer for the previous query. There are three possible commands:

- **?**
- **! x y**
- **end**

For each query, you are supposed to ask for the task's coordinates by using the **?** command and then report your answer by using the **!** command. Any other variation of commands will result in a WA. The answer is not valid if the server and/or CPU doesn't exist or if that CPU is busy at that time. The communication **must** end with a call to the **end** command. The queries are separated by 1 unit of time, and they should be scheduled immediately.

Note: Please end any command with a '\n' and remember to flush the buffers.

Scoring

Your score for one test is the sum of costs for all the queries (using the cost measure from the problem's description). The final score is the sum of scores in all tests. The goal is to ***minimize*** that score.

If your program works incorrectly (e.g. it exceeds the time limit or the answer is not valid) on any of the tests, you will get a suitable verdict (e.g. TLE or WA). Otherwise, you will get AC and your score will be decided by only a part of the tests (see test generation). The final score will be revealed after the contest.

Constraints

- $\mathbf{1 \le N \le 100{,}000}$
- $\mathbf{1 \le Q \le 100{,}000}$
- The total number of CPUs is at most **500,000**.
- A server can have no CPUs. That is, **k** can be 0.
- All servers and CPUs are indexed from **1**.
- All coordinates are between **0** and **100,000**.
- All processing times are between **0** and **85,000**.
- The tasks and servers may overlap (two tasks or a task and a server or two servers can share coordinates).
- The first task arrives at time **0** (the second one is at time 1 and so on).
- It is guaranteed that the number of tasks is not greater than the number of cores (i.e. there is no strategy that leaves you with no available CPU for some task).
- If a task is executed at time **t** on a CPU of processing time **p** then the CPU will be busy in the time interval **[t, t+1, ..., t+p-1].** (the CPU becomes available at time **t+p**).
- A task cannot be stopped or rescheduled.
- **Note**: there is **no** actual time-stamp associated with a task. The times are introduced so it is easier to express the availability of a CPU and the fact that tasks arrive one after the other (separated by one unit of time). The judge also uses time **0** as the original time in order to check if a schedule is valid.
- **Note**: We use the **classical euclidean distance**: $euclid\_distance((a, b), (x, y)) = sqrt((a - x)^2 + (b - y)^2)$
- **Note**: The TL is given for one test and not for a single interaction (i.e. single message exchange).

Test generation

There are several types of datasets:

- Dataset #0: single server

    - **N = 1**

- Dataset #1: small network
    - **2 ≤ N ≤ 100**

- Dataset #2: small number of tasks
    - **Q ≤ 1000**

- Dataset #3: fast network
    - All processing times ≤ **1000**

- Dataset #4: small Batcave
    - All coordinates ≤ **3000**

- Dataset #5: random network and testing framework

Distribution (20 tests):

- single server: 3 tests
- small network: 2 tests
- small number of tasks: 2 tests
- fast network: 3 tests
- small Batcave: 4 tests
- random network: 6 tests

Note: During the contest your code will run against only one test from each type of dataset.

Example

```
Input:
2 3
9 9 2 7 9
0 0 2 10 5
0 2
5 7
8 3

Output:
2 2
1 1
1 2
```

Explanation

There are 3 tasks that are given as 2D points (0,2), (5,7) and (8,3).

There are 2 servers, each having 2 CPUs.

The first server has two CPUs of times 7 and 9.

The second server has two CPUs of times 10 and 5.

The user should first read the description of the servers from standard input (as mentioned in the Input section). Then, the interaction between the two programs (3 bi-directional message exchanges and the end command) should be:

```
User             Judge

?
                      0 2
! 2 2
?
                      5 7
! 1 1
?
                      8 3
! 1 2
end
```

The cost of the first query: sqrt((0 - 0)^2 + (0 - 2)^2) + 5 = sqrt(4) + 5 = 7

The cost of the second query: sqrt((9 - 5)^2 + (9 - 7)^2) + 7 = sqrt(4^2 + 2^2) + 7 = sqrt(16 + 4) + 7 = 11.472135

The cost of the third query: sqrt((9 - 8)^2 + (9 - 3)^2) + 9 = sqrt(1^2 + 6^2) + 9 = sqrt(1 + 36) + 9 = 15.082762

The score for this test is the sum of all costs: 7 + 11.472135 + 15.082762 = 33.554897

Note: the answer is clearly not unique and in this case is not intended to be optimal.

Author:	3★ alexvaleanu (/users/alexvaleanu)
Date Added:	4-07-2017
Time Limit:	1.5 secs
Source Limit:	50000 Bytes
Languages:	ADA, ASM, BASH, BF, C, C99 strict, CAML, CLOJ, CLPS, CPP 4.3.2, CPP 4.9.2, CPP14, CS2, D, ERL, FORT, FS, GO, HASK, ICK, ICON, JAVA, JS, LISP clisp, LISP sbcl, LUA, NEM, NICE, NODEJS, PAS fpc, PAS gpc, PERL, PERL6, PHP, PIKE, PRLG, PYPY, PYTH, PYTH 3.4, RUBY, SCALA, SCM chicken, SCM guile, SCM qobi, ST, TCL, TEXT, WSPC

Submit (/JULY17/submit/SRVRS)

Comments ▸