A. BuberPool Taxi Optimization

time limit per test: 15 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

This is an interactive optimization problem. Instead of having to find the best answer, each solution will get score according to its efficiency. As the problem is interactive, during the testing process, the jury program and your solution will exchange messages via standard input and output streams. After printing a message, remember to flush the output buffer to ensure that the message is fully sent. For example, you can use `fflush(stdout)` in C++, `System.out.flush()` in Java, `flush(output)` in Pascal, and `stdout.flush()` in Python.

Your task is to automate the next generation taxi service, BuberPool. Each Buber car may transport up to four passengers at a time. The cars have to drive to passengers' locations, pick them up and drive them to their destinations. Additionally, unlike a traditional taxi, each car can process up to four orders simultaneously. For example, when driving a passenger to the destination, a car can pick up an additional passenger, or make a detour to drop off another passenger.

We will use the following simplified discrete model for this real-life problem.

Consider a city as a rectangular grid of w streets and h avenues with $w \times h$ crossroads. The streets are numbered from 1 to w, and the avenues from 1 to h. We will consider non-negative integer moments of time. Actions are performed in ticks: intervals of time between consecutive moments. At any moment, each car is located at some crossroads (x, y), where $1 \le x \le w$ and $1 \le y \le h$. Any number of cars can be at the same crossroads simultaneously.
During one tick, each car can either remain on the same crossroads or change exactly one of its coordinates, the street or the avenue, by one.

At moment 0, you are given starting locations of k cars: (x_1, y_1), (x_2, y_2), …, (x_k, y_k). The number of cars is equal to k and does not change during the simulation.

Additionally, your solution will be given the clients' orders. In this problem, each order is made by a single passenger, so a car can process up to four orders at the same time. To further simplify the problem, the moments of orders are all distinct, and the orders are given in chronological order. So, each j-th order is described by five integers: the moment t_j, the pick up location (sx_j, sy_j), and the drop-off location (tx_j, ty_j).

On tick 0, and also right after receiving each order, you can give instruction sets to the car drivers. Each time you can give instruction sets to any number of cars (more on restrictions later). Instruction sets for different cars are given separately. An instruction set is a sequence of triples (cx_1, cy_1, a_1), (cx_2, cy_2, a_2), …, (cx_m, cy_m, a_m), where m is number of instructions in the set. The meaning of these numbers is as follows:

- A car should first drive to (cx_1, cy_1), then to (cx_2, cy_2), …, and at last to (cx_m, cy_m). Each location (cx_i, cy_i) may be any crossroads in the city, not necessarily a pickup or drop-off location. While driving from one location to another, a car always first changes its first coordinate until it is equal to the first coordinate of the destination, and only then the second coordinate. This way, while driving from (p_1, q_1) to (p_2, q_2), a car will first move $(p_1, q_1) \to (p_2, q_1)$, and then $(p_2, q_1) \to (p_2, q_2)$. In each tick while moving, a car changes exactly one of its coordinates by one.
- Values a_i encode actions. If $a_i = 0$, the car just drives to (cx_i, cy_i) and doesn't do anything special. If $a_i > 0$, the car drives to (cx_i, cy_i) and picks up passenger number a_i there (the passenger's pickup location should be equal to (cx_i, cy_i), and the passenger should still be waiting there). If $a_i < 0$, the car moves to (cx_i, cy_i) and drops off passenger number $-a_i$ (the passenger's drop-off location should be equal to (cx_i, cy_i), and the passenger should be in this car). Picking up and dropping off passengers are instant.

When you give instruction set to a car, its previous instruction set is replaced with the new one. This way, a car only performs at most one instruction set at a time. If there are no instructions, or all of them are completed, the car is waiting at its last location. Instructions are performed in the given order.

For each order, first, all the cars perform their current instruction sets until the moment of the order comes, then the order is given to your program, and after that, your program prints new instruction sets for some of the cars. The cars which don't get new instruction sets continue performing their old ones.

So, the overall testing strategy looks like this:

1. Your solution reads the city dimensions, the number of cars, and their initial coordinates.
2. The solution prints instruction sets for some cars, and the cars start performing the instructions.
3. The solution reads an integer t_j, the moment of the next order (if $t_j = -1$, your program should print the final instruction sets and exit normally).
4. The solution reads integers sx_j, sy_j, tx_j, ty_j, the pickup and drop-off locations for j-th order.
5. The solution prints the next instruction sets for the cars.
6. Go to step 3.

After your program exits in step 3, first, all instructions will be completely performed, and only after that the simulation will end.

The score your solution gets for a test is the average score for all orders, rounded to the nearest integer. The score for an order equals $\alpha \cdot (100 + w_0)$, where w_0 is time the order takes in the ideal case, which is just the time to get from (sx_j, sy_j) to (tx_j, ty_j) (so, $w_0 = |sx_j - tx_j| + |sy_j - ty_j|$), and α is a real value between 0 and 1 which characterizes the passenger's satisfaction with the ride. The value α is calculated by the formula:

$$\alpha = 10^7 - \min(d_{21} + d_{22}, 10^7)10^7,$$

where d_1 is the waiting time before pickup (the number of ticks between the time of the order and pickup time), and d_2 is the difference between actual and ideal travel time (the ideal travel time is w_0). If the order was not completed, $\alpha = 0$.

The score your solution gets is just the sum of scores for each test.

Input

The first line of input contains integers w and h ($300 \le w, h \le 3000$), the dimensions of the rectangular city. The second line contains an integer k ($1 \le k \le 40$), the number of cars. Next k lines contain integer coordinates x_i and y_i ($1 \le x_i \le w$, $1 \le y_i \le h$), the initial locations of cars. Recall that any number of cars can share the same location.

After reading this data, your program should output the initial instruction set for the cars (see output format), and then process the orders.

Each order is given on its own line and is described by five integers t_j, sx_j, sy_j, tx_j, ty_j ($1 \le t_j \le 86400$, $1 \le sx_j, tx_j \le w$, $1 \le sy_j, ty_j \le h$), where t_j is the moment of the order, (sx_j, sy_j) is the passenger's pickup location and (tx_j, ty_j) is the passenger's drop-off location. Orders are sorted by strictly increasing t_j. In each order, the pickup and drop-off locations are different.

After all orders, your program is given a line with five integers -1. In other words, if after reading another order, $t_j = -1$, it means that there are no more orders. It is guaranteed that the number of orders is between 1 and 500.

After reading each order, print the instruction sets for cars. Also, print the instruction sets after reaching the line with five integers -1. So, if the input contains q orders, then the number of times to print instruction sets for cars is $q + 2$ (the initial one, one after each order, and the final one).

Remember that simulation is implemented in such a way that, for each order, first, all the cars perform their current instruction sets until the moment of the order comes, then the order is given to your program, and after that, your program prints new instruction sets for some of the cars. The cars which don't get new instruction sets continue performing their old ones.

During main part of the contest your solution will be tested on preliminary testset of 40 tests. Half of these tests (tests with odd numbers) are available for download at this url: https://assets.codeforces.com/rounds/927/tests.zip. There is a test generated with each generation method prepared by the jury among the published tests.

After the round, last solution that has positive score on preliminary test will be used for testing. Other solutions will be ignored. All chosen solutions will be tested on secret testset, that is generated the same way (same generation methods), as preliminary tests. Jury will try to save test types and not to change statistic parameters a lot.

Output

The output of your solution is instruction sets for the cars. Carefully read the problem statement and input format to understand the simulation process. Each of the $q + 2$ messages you print should be a sequence of integers on a single line.

Each message starts with a number f ($0 \le f \le k$), that number of cars that will receive new instructions. It should be followed by f space separated blocks.

Each block corresponds to an instruction set for a single car. A block starts with a pair of integers c and m ($1 \le c \le k$), the number of the car and the length of the instruction sequence for this car. Next there should be m instructions: integer triples cx, cy, a ($1 \le cx \le w$, $1 \le cy \le h$, $-q \le a \le q$), where a triple means that the car should drive to location (cx, cy) and perform action a (read the main part of the statement). Here, q is the number of orders received so far.

There should be no more than 10^6 instructions in all of the output of your program.

After printing the instruction sets, but before waiting for the next input, remember to flush the output buffer to ensure that the message is fully sent. For example, you can use `fflush(stdout)` in C++, `System.out.flush()` in Java, `flush(output)` in Pascal, and `stdout.flush()` in Python.