2009 Canadian Computing Competition
Day 2, Question 1

Problem D – Bottle Caps

Input: none
Output: none
Source file: `bottlecap.{c, cpp, pas}`

You have N ($1 \leq N \leq 10,000$) caps and N bottles. Each bottle has a unique size and each cap corresponds to a single bottle. Unfortunately, the caps and bottles are so close together in size that it is impossible to compare them visually.

The only comparison that we can make is between a cap and a bottle. We do this by attempting to put the cap on the bottle. From this, we know whether the cap is too small, too large, or just the right size in relation to the bottle.

Match all the caps to their corresponding bottles as quickly as possible by interacting with a library that compares the pieces.

You may make at most $500,000$ queries.

In 20% of the test cases, $N \leq 700$. In another 40% of the test cases, $N \leq 3000$.

Interactive Instructions

If you are using C/C++, you need to include the header file `bottlecaplib.h` in your source code and compile your program with the supplied library (using `g++ -o bottlecap bottlecaplib.o bottlecap.cpp` or `gcc` instead of `g++`).

If you are using Pascal, your program must contain the statement `uses bottlecaplib;`.

There are 3 functions to call, which are:

- `int getN()` in C/C++ or
  `function getN():integer` in Pascal

  This function returns the value of N and should be called exactly once before calling any other functions.

- `int query(int capid,int bottleid)` in C/C++ or
  `function query(capid:integer, bottleid:integer):integer` in Pascal

  This is the query function. The function returns 1 if the cap too large for the bottle, 0 if the cap fits the bottle, and -1 if the cap is too small for the bottle.

- `void report(int capid,int bottleid)` in C/C++ or
  `procedure guess(capid:integer, bottleid:integer)` in Pascal

  This method reports that the cap with id `capid` maps to the bottle with id `bottleid`. Your program should report once for each cap/bottle; any duplicate reporting will result in 0 points. Once you have reported all the caps, the library will terminate your program automatically. Your program should NEVER quit on its own.

Testing

To test your program, create an input file called `bottlecap.in` as described below. The supplied library (and the library we will test your program with) will read the input file and perform the comparisons for you while keeping track of the number of performed queries. The sequence of calls made to the library by your program and correctness of the reports will be displayed on standard output.

Do not try to access the variable names given in the supplied library as it won't work in the actual testing library.

Input Specification (for testing)

The first line contains the integer N.

The next N lines contain a permutation of the integers from 1 to N (inclusive), with one integer per line, representing the size of the caps (with line i in this set representing the size of cap i).

The next N lines contain a permutation of the integers from 1 to N (inclusive), with one integer per line, representing the size of the bottles (with line j in this set representing the size of cap j).

Sample Input (for testing)

```
3
2
1
3
3
2
1
```

Description of Sample Input (for testing)

There are three caps and three bottles. The first, second and third cap match the second, third, and first bottle, respectively.

Output Specification (for testing)

The output to standard output contains descriptions of the calls to the library routines. The last line of output will be either `correct` or one of many other informative error messages (such as `Mismatch` or `Duplicated report`).

2009 Canadian Computing Competition
Day 2, Question 2

Problem E – Parade

Input: standard input
Output: standard output
Source file: `parade.{c, cpp, pas}`

May 9 is VE day, when the annual victory parade is held through Red Square. Inspired by the award ceremony of the 2009 ACM World Finals, you have decided to recreate the original VE day parade digitally. Since you have skillfully obtained all the necessary hardware, the most difficult part remaining is to track the configuration of a single formation.

A formation can be viewed as a 4-by-4 array, with the people initially labelled 1 through 16:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Then a sequence of commands (r, c, k) are issued. Each command is a "rotation" based on the three integer parameters r, c and k:

> Rotate all people on the perimeter of a k-by-k square with its upper left corner located at row r, column c clockwise by one position.

For example, the command $(1, 1, 2)$ would alter the initial board to:

5	1	3	4
6	2	7	8
9	10	11	12
13	14	15	16

As another example, $(2, 2, 3)$ would bring the initial board to

1	2	3	4
5	10	6	7
9	14	11	8
13	15	16	12

A third example, $(1, 1, 4)$ would bring the initial board to

5	1	2	3
9	6	7	4
13	10	11	8
14	15	16	12

You have obtained the original sequence of commands issued. However, you are not quite pleased with the final result and would like to edit some of the commands. Support Q ($1 \leq Q \leq 100,000$) changes of the command sequence, each change as follows:

Change the ith command to r', c' and k' permanently.

Furthermore, you would like to see the effects of your change immediately. After each change, output what the formation would looks like at the end of *all* N commands. To re-emphasize, each change is cumulative and permanent.

For 30% of the test cases, $1 \leq N, Q \leq 1000$.

Input Specification

The first line contains two integers N and Q ($1 \leq N, Q \leq 100,000$), which is the number of commands and number of edits, respectively.

The next N lines contain three integers per line: r c and k which describe each rotation command. Note that $1 \leq k \leq 4$, $r + k - 1 \leq 4$ and $c + k - 1 \leq 4$.

The next Q lines contain four integers per line: the first integer on the line is the 1-based index of the command whose detail is to be changed, followed by 3 integers, r', c', k', the new description of the command.

Sample Input

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

Output Specification

For each change, print 4 lines, the final configuration of the formation after the modifications so far.

Output for Sample Input

```
5 1 3 4
6 2 7 8
9 10 11 12
13 14 15 16
5 1 3 4
6 10 2 7
9 14 11 8
13 15 16 12
```

```
1 2 3 4
5 10 6 7
9 14 11 8
13 15 16 12
5 1 2 3
9 6 7 4
13 10 11 8
14 15 16 12
```

Description of Output for Sample Input

The two commands $(1, 1, 1)$ leave the configuration unchanged. The first change $(1, 1, 2)$ on the initial configuration causes the configuration to become the first configuration in this problem statement. That is,

$$\begin{array}{cccc} 5 & 1 & 3 & 4 \\ 6 & 2 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{array}$$

The second change takes this configuration and applies $(2, 2, 3)$.

The next change alters the first change to be the "original" graph, and then, since the second command has been changed to $(2, 2, 3)$, we have

$$\begin{array}{cccc} 1 & 2 & 3 & 4 \\ 5 & 10 & 6 & 7 \\ 9 & 14 & 11 & 8 \\ 13 & 15 & 16 & 12 \end{array}$$

The last change causes the second command to be $(1, 1, 4)$ which rotates the outermost perimeter of the previous output.

2009 Canadian Computing Competition
Day 2, Question 3

Problem F – A Weighty Problem

Input: standard input
Output: standard output
Source file: `weight.{c, cpp, pas}`

You like to make purchases using coins, but you have a problem: you have so much change that it is too heavy in your pocket. You have devised a plan to reduce the weight of your change, and you need to write a program to help you execute it.

Here is your plan. The next purchase you make costs C ($1 \leq C \leq 100,000$) cents, and you know how the store will pay back change if you pay extra. You want to select some of your coins that have total value at least C and make the purchase, such that you minimize the weight of the coins you did not spend added to the weight of the coins the store returns to you.

If the store owes you X cents, then it uses the following algorithm to pay you back. The store gives you the largest denomination coin that has value at most X, and repeats this until all X cents have been paid to you. You may assume the store has an unlimited amount of every denomination of coin.

There are D ($1 \leq D \leq 100$) denominations of coins. The i'th denomination ($1 \leq i \leq D$) has integer value V_i ($1 \leq V_i \leq 2000$) in cents and real-valued weight W_i ($0 < W_i < 10$) in grams. You may assume that one of the denominations has value 1 and that no two denominations have the same value.

You have K ($1 \leq K \leq 100$) coins. The j'th coin ($1 \leq j \leq K$) is of the denomination with index D_j ($1 \leq D_j \leq D$).

In 20% of test cases, $K \leq 10$.

Input Specification

The first line contains three integers: C, the cost of the purchase in cents; D, the number of denominations of coins; and K, the number of coins you have.

The next D lines contain an integer V_i, the value of the i'th denomination in cents, and a real number given to two decimal places, W_i, the weight of the i'th denomination in grams.

The next K lines contain an integer D_j, the 1-based denomination of the j'th coin you have.

Sample Input

```
3 4 7
1 1.00
5 2.00
20 9.00
```

```
10 1.00
2
2
2
2
2
2
2
```

Description of Sample Input

You have seven 5-cent coins, and are making a purchase of 3 cents. The denominations are 1-cent, 5-cents, 10-cents, and 20-cents, with respective weights of 1 gram, 2 grams, 1 gram and 9 grams.

Output Specification

Output the minimum weight achievable rounded to two decimal places, if you can afford the purchase. If you cannot afford the purchase, print `too poor`.

Output for Sample Input

```
11.00
```

Description of Output for Sample Input

There are two optimal solutions. One is to spend three 5-cent coins, so that the store returns 12 cents to you, in the form of one 10-cent coin and two 1-cent coins. The other is to spend four 5-cent coins, so that the store returns 17 cents to you, in the form of one 10-cent coin, one 5-cent coin, and two 1-cent coins.