Problem A
Cardiology
Time limit: 2 seconds

The Great Cardoni, Master Prestidigitator, has a deck of 21 numbered cards which he uses in a trick as follows:

> A spectator secretly selects a number between 1 and 21, inclusive, after which Cardoni deals the 21 cards, face-up, row by row in order from 1 to 21, into a 3-column grid. The spectator then indicates which of the three columns contains the selected card, at which point the magician picks up the cards by columns, picking up the specified column second (the order of collecting the other two columns is unimportant). Cards are collected face up, beginning with the top card in each column and placing each succeeding card immediately beneath the previously collected card. The cards are then redealt by rows into a 3-column grid, starting from the top of the face-up deck. The process is repeated two more times; each time, the column indicated by the spectator is the second column picked up by the magician. After three such iterations, Cardoni announces, "I have penetrated to the heart of your mind; your card lies at the heart of this display." And it's true—the selected card is located at the "heart" of the array (row four, column two). Moreover, the selected card will always remain in this stable location for any further iterations of the column indication and card redealing process.

The process always works, no matter the number selected, provided that the column containing the secret number is the second column to be picked up and redealt.

Cardoni would like to expand his trick to use different numbers of cards, rows, and columns, and to experiment with different orderings of picking up the columns after the spectator indicates a column. However, it is not a trivial problem. For instance, when using 24 cards in 8 rows and 3 columns, and always picking up the indicated column as the second one to redeal, the number 5 eventually ends up in stable location row 4, column 3, while the number 20 ends up in stable location row 5, column 1. Also, neither location is one of the two "heart" positions in column 2 of rows 4 or 5. Moreover, Cardoni is uncertain of how many iterations of the "indicate column and redeal" process are needed before a selected card reaches a stable location.

Given the number of rows and columns of cards, help Cardoni set up his trick in such a way that there is a unique stable location that is as close to the center as possible.

Input

The input consists of a single line with two integers r and c ($2 \leq r, c \leq 10^6$), the number of rows and columns used in the trick. The cards are numbered from 1 to $r \cdot c$ and are initially dealt row by row in increasing order.

Output

Output a line containing four integers p, i, j, and s, where:

- the column indicated by the spectator should be picked up as the p^{th} column,
- using this value of p causes all cards to eventually end up in the stable location at row i column j, and
- s is the maximum number of iterations required for any card to reach the stable location.

The value of p should be chosen so that the stable location (i, j) is as close as possible to any of the one, two or four central positions in the grid, where the distance between locations (i, j) and (i', j') is $|i - i'| + |j - j'|$. If more than one value of p results in the same minimum distance, choose the smallest such p.

Sample Input 1	Sample Output 1
7 3	2 4 2 3

Sample Input 2	Sample Output 2
8 3	1 1 1 3

Problem B
The Cost of Speed Limits
Time limit: 8 seconds

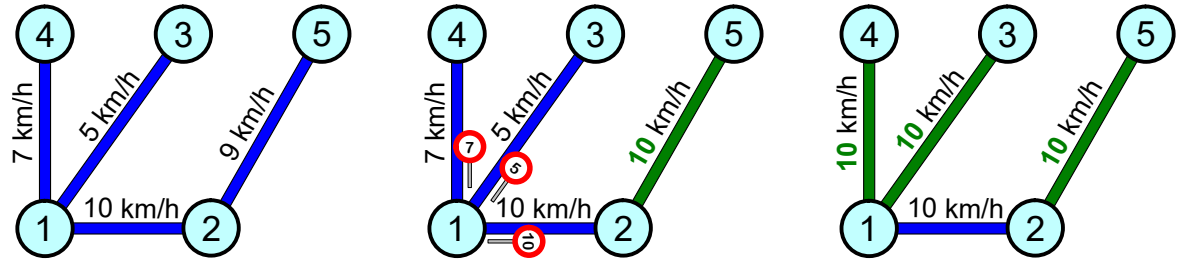
By the year 3031, the ICPC has become so popular that a whole new town has to be built to house all the World Finals teams. The town is beautifully designed, complete with a road network. Unfortunately, when preparing the budget, the town planners forgot to take into account the cost of speed-limit signs. They have asked you to help them determine the minimum additional funds they will need.

The ICPC road network consists of roads, each connecting two intersections. Each road is two-way and has already been assigned a speed limit, valid for both directions. To save money, the minimum possible number of roads was used. In other words, there is exactly one route from any intersection to any other intersection.

The speed-limit signs need to be installed in all places where the speed limit may change for any driver that follows any route. More precisely, if there exists an intersection where at least two roads meet with different speed limits, then *all* of the roads going from that intersection need a speed-limit sign installed at that intersection. Note that some roads might need two speed-limit signs, one at each end.

It costs c dollars to install one speed-limit sign. It is also possible to improve the safety and quality of any road so that its speed limit can be increased, which may in turn reduce the number of speed-limit signs required. It costs x dollars to increase the speed limit of one road by x km/h (in both directions). To avoid complaints, the town council does not allow decreasing any of the already-assigned speed limits.

Figure B.1 illustrates the situation given in both Sample Input 1 and Sample Input 2.



(a) Town roads with originally assigned speed limits.

(b) The solution to Sample Input 1 involves installing three signs and upgrading one road.

(c) If c is too high, it is possible to upgrade roads instead, so all of them have the same speed limit. Then we need no signs, as in Sample Input 2.

Figure B.1: Illustration of the road network and speed limits.

Input

The first line of input contains two integers n and c, where n ($1 \leq n \leq 20\,000$) is the number of intersections and c ($1 \leq c \leq 10^5$) is the cost of installing one sign. Each of the remaining $n-1$ lines contains three integers u, v, and s, where u and v ($1 \leq u, v \leq n; u \neq v$) are the intersections at the ends of a road, and s ($1 \leq s \leq 10^5$) is the current speed limit of that road in kilometers per hour.

Output

Output the minimum cost to upgrade roads and install speed-limit signs such that the town plan satisfies all the rules above.

Sample Input 1	Sample Output 1
5 2 1 2 10 1 3 5 1 4 7 2 5 9	7

Sample Input 2	Sample Output 2
5 100 1 2 10 1 3 5 1 4 7 2 5 9	9

Problem C
Domes
Time limit: 2 seconds

Saint Basil's Cathedral is the best-known landmark of Moscow and maybe even of all of Russia. Built under Ivan the Terrible in the 16th century, the cathedral is known for its colorful domes. No visit to the city is complete without taking a photo of the former church in Red Square.

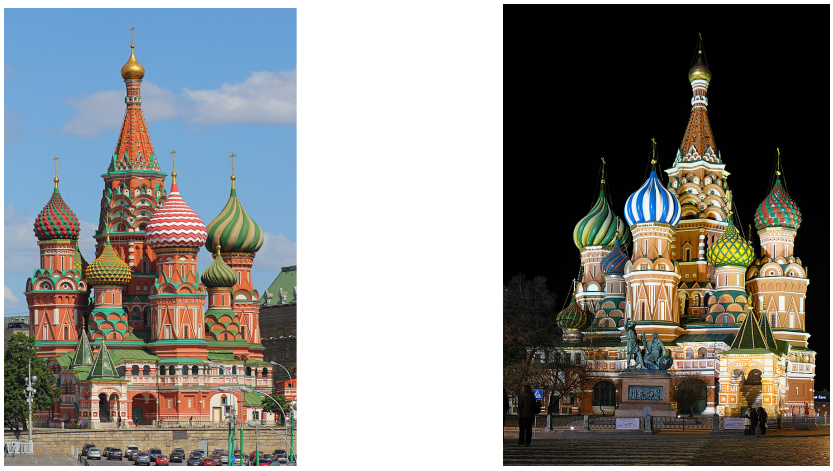


Figure C.1: Two views of Saint Basil's Cathedral.

The Moscow Tourism Board (MTB) wants to make it as safe as possible for tourists to take the perfect shot of the cathedral. Depending on where you stand when you take a picture, the relative positions of the domes will be different (see Figure C.1). The MTB is concerned that for some desired configurations of domes the region in Red Square where such a photo is possible will be so small as to lead to a dangerous overcrowding of photographers. Wanting to avoid the inevitable pushing, shoving, injury, and Covid that this could cause, the MTB would like to find the area of the region where a photo is possible for any desired ordering of the domes.

For simplicity, assume that cameras have a 180-degree viewing angle. As an illustration consider Figure C.2, which shows the location of the domes (labeled 1–5) and the photographer (green dot) in the plane. If the photographer shoots a picture aiming the camera straight towards the left (directly at dome 5), then everything in the shaded area will be visible in the photograph. Note that in this photograph, the domes will appear in order $4, 3, 5, 2, 1$ from the left to the right.

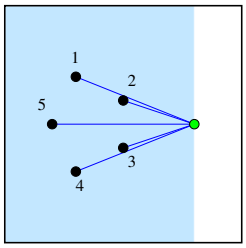


Figure C.2: Illustration of the sample inputs.

Given the location of the domes within Red Square and a desired left-to-right order of the domes in the photograph, MTB wants to know the area of the region within Red Square from which such a photograph can be taken. You can assume that the domes are points, so that they do not block each other unless they are in a straight line from the photographer's view.

Input

The first line of input contains three integers d_x, d_y, and n, where d_x and d_y $(2 \leq d_x, d_y \leq 10^5)$ are the dimensions of Red Square, and n $(1 \leq n \leq 100)$ is the number of domes. The bottom-left corner of Red Square is at the origin $(0, 0)$ and the top-right corner is at coordinate (d_x, d_y).

Each of the next n lines contains two integers x_i and y_i $(0 < x_i < d_x, 0 < y_i < d_y)$, giving the locations (x_i, y_i) of the domes. The i^{th} line describes dome number i. No two domes are in the same location.

The last line contains a permutation of the numbers $\{1, \ldots, n\}$ specifying the desired left-to-right viewing order of the domes in the picture.

Output

Output the area of the region within Red Square from which one can take a photo that shows the domes in the requested order. Note that the area may be 0 if there is no position from which to take the requested photo. Your answer should have an absolute or relative error of at most 10^{-3}.

Sample Input 1

```
100 100 5
30 70
50 60
50 40
30 30
20 50
4 3 5 2 1
```

Sample Output 1

```
450.000000
```

Sample Input 2

```
100 100 5
30 70
50 60
50 40
30 30
20 50
1 2 5 4 3
```

Sample Output 2

```
0.000000
```

Problem D
Gene Folding
Time limit: 5 seconds

International Cell Processing Company (ICPC) is a world leader in the analysis of genetic sequences. A genetic sequence is a sequence of nucleotides, which in this problem is represented by a string containing only the letters `A`, `C`, `G`, and `T` in some combination, each letter representing a single nucleotide (**A**denine, **C**ytosine, **G**uanine, and **T**hymine, respectively).

One of the key discoveries made by ICPC is that through a process called Genetically Optimized Organic Folding (GOOF), they can take a genetic sequence and transform it into a simpler one, while preserving many of the properties of the sequence that ICPC wants to analyze.

A single application of GOOF works as follows. Find a point between two adjacent nucleotides in the nucleotide sequence, such that the sequence reads the same from that point in both directions, up until the nearer end of the sequence. For instance, in the sequence `ATTACC`, there are two such points: `AT-TACC` and `ATTAC-C`. Then pick one of these points (say, the first one), and fold the genetic sequence at that point, merging the identical nucleotides (so, in this case the `AT` and `TA` would become merged, and the resulting sequence would be `CCAT` or `TACC`).

Through repeated application of GOOF, a nucleotide can potentially be made much shorter. However, manually searching for the appropriate folding points is very time-consuming. ICPC reached out to you to write a program that would automate the process of finding the folding points and choosing them so as to obtain the shortest possible genetic sequence from a given input sequence.

Input

The input contains a single string s representing the nucleotide sequence to be analyzed. The string consists of characters `A`, `C`, `G`, and `T` only. The length of s is between 1 and $4 \cdot 10^6$, inclusive.

Output

Output the smallest possible length of a sequence obtained from the input by applying GOOF zero or more times.

Sample Input 1	Sample Output 1
ATTACC	3

Sample Input 2	Sample Output 2
AAAAGAATTAA	5

This page is intentionally left blank.

Problem E
Landscape Generator
Time limit: 4 seconds

Interactive Creative Players Collective (ICPC) is working on a new computer game for which they want to generate realistic landscapes. One of the ICPC engineers proposed an algorithm inspired by geological processes. The algorithm starts with a flat landscape and repeatedly modifies it by lifting or lowering continuous blocks, thus forming *horsts* (lifted blocks) and *grabens* (lowered blocks). The blocks to be lifted or lowered are selected at random. ICPC hopes to obtain realistic landscapes this way.

Your task is to interpret any sequence of such modifications and output the resulting landscape. The landscape is represented by a sequence of n integer height values, one for each integer point from 1 to n on the x-axis. Figure E.1 illustrates an example by connecting the height values with line segments.

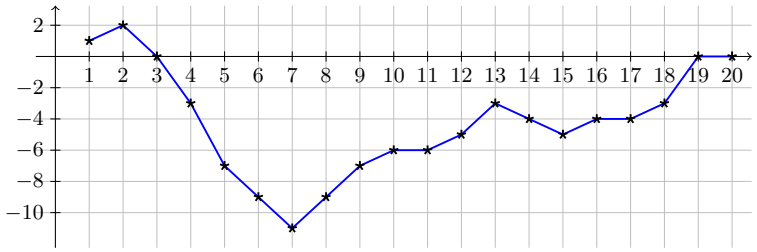


Figure E.1: Illustration of the landscape generated by Sample Input 1.

Initially the height is 0 at all n points. This flat shape is subjected to a sequence of modifications. Each modification applies one of the following four operations with two integer parameters $x_1 \leq x_2$:

- `R`: Raise – increase the height by 1 at all points between x_1 and x_2 inclusive.
- `D`: Depress – decrease the height by 1 at all points between x_1 and x_2 inclusive.
- `H`: Hill – add a new linearly shaped hill between x_1 and x_2.
- `V`: Valley – add a new linearly shaped valley between x_1 and x_2.

Adding a hill to the current landscape works as follows. The heights at points x_1 and x_2 are increased by 1. If $x_2 - x_1 > 1$, the heights at points $x_1 + 1$ and $x_2 - 1$ are increased by 2. If $x_2 - x_1 > 3$, the heights at points $x_1 + 2$ and $x_2 - 2$ are increased by 3, and so on. Figure E.2 shows an example. Adding a valley works in the same way except the heights are decreased instead. The maximal change of height happens in the middle between x_1 and x_2. If $x_2 - x_1$ is odd, there will be two neighboring points with maximal change, otherwise just one.

Input

The first line of input contains two integers n and k, where n ($1 \leq n \leq 200\,000$) is the number of points, and k ($0 \leq k \leq 200\,000$) is the number of modifications. The n points along the x-axis are numbered from 1 to n. The next k lines describe the modifications. Each line contains one character c and two integers x_1 and x_2, where c (one of `R`, `D`, `H` or `V`) designates the operation and x_1 and x_2 ($1 \leq x_1 \leq x_2 \leq n$) specify its parameters.

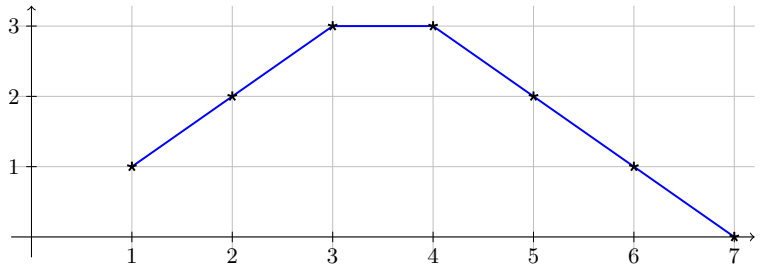


Figure E.2: Illustration of the landscape generated by Sample Input 2.

Output

Output n lines, where the i^{th} line contains the height at point i after applying all modifications in the given order.

Sample Input 1	Sample Output 1
20 13 H 12 13 D 5 18 R 13 14 R 8 16 H 2 3 V 10 19 V 3 13 R 8 13 V 3 10 D 5 18 V 11 12 R 1 6 R 14 19	1 2 0 -3 -7 -9 -11 -9 -7 -6 -6 -5 -3 -4 -5 -4 -4 -3 0 0

Sample Input 2	Sample Output 2
7 1 H 1 6	1 2 3 3 2 1 0

Problem F
Ley Lines
Time limit: 15 seconds

In 1921, the amateur archaeologist Alfred Watkins coined the term "ley lines" to refer to straight lines between numerous places of geographical and historical interest. These lines have often been associated with mysterious and mystical theories, many of which still persist.

One of the common criticisms of ley lines is that lines one draws on a map are actually of non-zero width, and finding "lines" that connect multiple places is trivial, given a sufficient density of points and a sufficiently thick pencil. In this problem you will explore that criticism.

For simplicity, we will ignore the curvature of the earth, and just assume we are dealing with a set of points on a plane, each of which has a unique (x, y) coordinate, and no three of which lie on a single straight line. Given such a set, and the thickness of your pencil, what is the largest number of points through which you can draw a single line?

Input

The first line of input consists of two integers n and t, where n ($3 \leq n \leq 3\,000$) is the number of points in the set and t ($0 \leq t \leq 10^9$) is the thickness of the pencil. Then follow n lines, each containing two integers x and y ($-10^9 \leq x, y \leq 10^9$), indicating the coordinates of a point in the set.

You may assume that the input is such that the answer would not change if the thickness t was increased or decreased by 10^{-2}, and that no three input points are collinear.

Output

Output the maximum number of points that lie on a single "line" of thickness t.

Sample Input 1	Sample Output 1
4 2 0 0 2 4 4 9 3 1	3

Sample Input 2	Sample Output 2
3 1 0 10 2000 10 1000 12	2

This page is intentionally left blank.

Problem G
Opportunity Cost
Time limit: 5 seconds

As with most types of products, buying a new phone can be difficult. One of the main challenges is that there are a lot of different aspects of the phone that you might care about, such as its price, its performance, and how user-friendly the phone is. Typically, there will be no single phone that is simultaneously the best at all of these things: the cheapest phone, the most powerful phone, and the most user-friendly phone will likely be different phones.

Thus when buying a phone, you are forced to make some sacrifices by balancing the different aspects you care about against each other and choosing the phone that achieves the best compromise (where "best" of course depends on what your priorities happen to be). One way of measuring this sacrifice is known as the *opportunity cost*, which (for the purposes of this problem) we define as follows.

Suppose that you have bought a phone with price x, performance y, and user-friendliness z. For simplicity, we assume that these three values are measured on a comparable numeric scale where higher is better. If there are n available phones, and the values (x_i, y_i, z_i) represent the (price, performance, user-friendliness) of the i^{th} phone, then the opportunity cost of your phone is defined as

$$\max_{1 \le i \le n} \left(\max(x_i - x, 0) + \max(y_i - y, 0) + \max(z_i - z, 0)\right).$$

Write a program that, given the list of available phones, finds a phone with the minimum opportunity cost.

Input

The first line of input contains an integer n ($2 \le n \le 200\,000$), the number of phones considered. Following that are n lines. The i^{th} of these lines contains three integers x_i, y_i, and z_i, where x_i is the price, y_i is the performance, and z_i is the user-friendliness of the i^{th} phone ($1 \le x_i, y_i, z_i \le 10^9$).

Output

Output a single line containing two integers: the smallest possible opportunity cost and an integer between 1 and n indicating the phone achieving that opportunity cost. If there are multiple such phones, output the one with the smallest index.

Sample Input 1	Sample Output 1
4 20 5 5 5 20 5 5 5 20 10 10 10	10 4

Sample Input 2	Sample Output 2
4 15 15 5 5 15 15 15 5 15 10 10 10	10 1

Problem H
QC QC
Time limit: 10 seconds

Innovative Computable Quality Control (ICQC) has developed a ground-breaking new machine for performing, well, quality control. Thanks to its novel Deep Intelligence technology, an ICQC quality control (QC) machine can automatically, with 100% accuracy, detect manufacturing errors in any machine in existence, whether it is a coffee machine, an intergalactic space ship, or a quantum computer.

ICQC is now setting up its factory for producing these QC machines. Like any other manufacturing process, some fraction of the produced machines will suffer from malfunctions and these need to be found and discarded. Fortunately, ICQC has just the product for detecting malfunctioning machines!

Obviously, ICQC should not simply use a QC machine on itself, since a malfunctioning machine might incorrectly classify itself as working correctly. Instead, ICQC will take each batch of n machines produced during a day and have them test each other overnight. In particular, during every hour of the night, each of the n QC machines can run a check on one of the other QC machines, and simultaneously be checked by one other QC machine.

If the machine running the check is correct, it will correctly report whether the tested machine is correct or malfunctioning, but if the machine running the check is malfunctioning, it may report either result. If a machine A is used to test a machine B multiple times it will return the same result every time, even if machine A is malfunctioning. The exact testing schedule does not have to be fixed in advance, so the choice of which machines should check which other machines during the second hour of the night may be based on the result of the tests from the first hour, and so on.

ICQC are 100% confident that strictly more than a half of the n QC machines in each batch are working correctly, but the night is only 12 hours long, so there is only time to do a small number of test rounds. Can you help ICQC determine which QC machines are malfunctioning?

For example, consider Sample Interaction 1 below. After the fourth hour, every machine has tested every other machine. For machine 1, only one other machine claimed that it was malfunctioning, and if it was truly malfunctioning then at least 3 of the other machines would claim this. For machine 4, only one other machine claims that it is working, which implies that machine 2 must be malfunctioning since more than half of the machines are supposed to be working. Note that even though machine 4 is malfunctioning, it still happened to produce the correct responses in these specific test rounds.

Interaction

The first line of input contains a single integer b ($1 \leq b \leq 500$), the number of batches to follow. Each batch is independent. You should process each batch interactively, which means the input you receive will depend on the previous output of your program.

The first line of input for each batch contains a single integer n ($1 \leq n \leq 100$), the number of QC machines in the batch. The interaction then proceeds in rounds. In each round, your program can schedule tests for the next hour, by writing a line of the form "`test` $x_1\ x_2\ \ldots\ x_n$" indicating that each machine i should run a test on machine x_i. If $x_i = 0$, then machine i is idle in that round and performs no test. All positive numbers in the sequence must be distinct.

After writing this line, there will be a result to read from the input. The result is one line containing

a string of length n, having a '1' in position i if machine i says that machine x_i is working correctly, '0' if machine i says that machine x_i is malfunctioning, and '–' (dash) if machine i was idle in the round.

When your program has determined which machines are malfunctioning, but no later than after 12 rounds of tests, it must write a line of the form "`answer` S" where S is a binary string of length n, having a '1' in position i if machine i is working correctly, and a '0' if it is malfunctioning.

After writing the answer line, your program should start processing the next batch by reading its number n. When all b batches have been processed, the interaction ends and your program should exit.

Notes on interactive judging:

- *The evaluation is non-adversarial, meaning that the result of each machine testing each other machine is chosen in advance rather than in response to your queries.*
- *Do not forget to flush output buffers after writing. See the Addendum to Judging Notes for details.*
- *You are provided with a command-line tool for local testing, together with input files corresponding to the sample interactions. The tool has comments at the top to explain its use.*

Sample Interaction 1

Read	Write
1 5	
	test 5 4 2 1 3
11011	
	test 4 5 1 3 2
00110	
	test 2 3 4 5 1
01011	
	test 3 1 5 2 4
10100	
	answer 10101

Sample Interaction 2

Read	Write
2 4	
	test 2 3 4 1
1111	
	answer 1111
7	
	test 2 3 4 5 6 7 1
0001100	
	test 0 0 0 0 2 4 0
----11-	
	answer 0101110

Problem I
Quests
Time limit: 10 seconds

To relax before competing in the ICPC World Finals, you have decided to play a computer game called *Quests*. You have played it a number of times already, and now you want to achieve a perfect playthrough—to prepare for your perfect playthrough of the finals!

In the game, you have to complete a number of quests, and you earn experience points (XPs) for completing each one. The total number of XPs you have earned at any time determines your current level. You reach a new level for every v XPs that you earn. Formally, your level at any time is the largest integer L such that you have at least $L \cdot v$ XPs.

Each quest is assigned a number x of XPs and a target difficulty level d. If you complete the quest when your level is at least d, you earn x XPs. However, if you complete the quest when your level is less than d, you will earn $c \cdot x$ XPs. The constant c is an XP multiplier that results in a bonus for completing a quest when you are at a lower level than the recommended level d.

You know all the n quests and their respective x and d numbers by heart (and you know the numbers v and c as well—you have played this game a lot). You are also skilled enough to complete any quest, regardless of its target difficulty level and your level. You want to complete all the quests in an order that will allow you to earn the largest possible number of XPs.

For example in the sample input, the maximum XPs you can earn is 43, which is done as follows. First complete the second quest (you earn 4 XPs, because you are at level 0, and you completed a quest with target difficulty level 2). Then complete the first quest (you earn 30 XPs, because you are still at level 0, and the target difficulty level is 1). With 34 XPs, you are now level 3. Finally, complete the third quest (for which you earn 9 XPs, without the multiplier, since you are already at level 3).

Input

The first line of input contains three integers n, v, and c, where n $(1 \leq n \leq 2\,000)$ is the number of quests in the game, v $(1 \leq v \leq 2\,000)$ is the number of XPs required to reach each level, and c $(2 \leq c \leq 2\,000)$ is the XP multiplier for completing a quest before reaching its target difficulty level.

Following that are n lines, each of which contains two integers x and d describing one quest, where x $(1 \leq x \leq 2\,000)$ is the number of XPs you earn for completing that quest if you are at or above its target difficulty level and d $(1 \leq d_i \leq 10^6)$ is the target difficulty level for that quest.

Output

Output the maximum possible number of XPs you could earn by finishing all the quests.

Sample Input 1	Sample Output 1
3 10 2 15 1 2 2 9 1	43

Problem J
'S No Problem
Time limit: 3 seconds

The Yllihc Engineering and Technological Institute (YETI), located in northern Snowblovia, has two problems: snow and money. Specifically, they have too much of the former and not enough of the latter. Every winter (and fall and spring, for that matter) the campus is covered with blankets of snow, and the sidewalks connecting campus buildings become impassable. To continue functioning, YETI needs to clear the snow from the sidewalks connecting the buildings of the campus. Since budget is an issue, these sidewalks are a minimal set that allow a path to exist between any two buildings.

With the money saved by not building more sidewalks, YETI bought two snow blowers. To clear the snow with them, two staff members take the two snow blowers out of the buildings (or building) they are stored in, and push them along the sidewalks, clearing the snow. Each sidewalk must be traversed at least once. Each snow blower, after it has finished, is stored in the building it is currently next to (and, during the next snowfall, the snow blowers will be pushed in the reverse direction—and so on, throughout the eleven months of the snow season).

The YETI maintenance crew want to choose the storage buildings of the snow blowers and design the routes along which they will be pushed, so that they minimize the total distance the two machines will travel out in the cold (to protect both the precious equipment and the staff members from freezing). Note that the routes might involve pushing along already-cleared sidewalks, as seen in Figure J.1, which shows an optimal solution for the sidewalk layout of Sample Input 1.

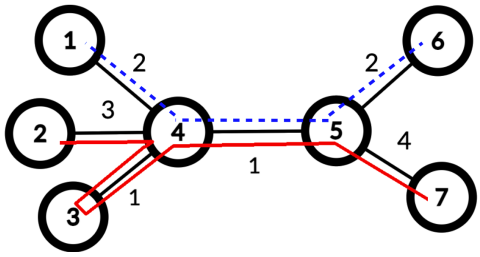


Figure J.1: Illustration of Sample Input 1 showing one possible pair of optimal routes.

YETI would ask their Computer Science Department to figure this out, but it was wiped out in the Great Blizzard of '06, so they have come to you for help.

Input

The first line of input contains an integer n ($4 \leq n \leq 100\,000$), the number of buildings on the YETI campus. Buildings are numbered from 1 to n. Each of the remaining $n-1$ lines contains three integers a, b, and d indicating that a sidewalk exists between buildings a and b ($1 \leq a, b \leq n$; $a \neq b$) of length d ($1 \leq d \leq 500$).

Output

Output the minimum total distance the snow blowers must travel to remove snow from all sidewalks.

Sample Input 1	Sample Output 1
7 1 4 2 2 4 3 3 4 1 4 5 1 5 6 2 5 7 4	15

Sample Input 2	Sample Output 2
4 1 2 1 2 3 2 3 4 3	6

Problem K
Space Walls
Time limit: 15 seconds

Place-Y Technology Corp. plans to launch a new space station soon. The company CEO is known for being obsessed with perfection. For example, he insists that all the outer surfaces of the space station are regularly polished and cleaned of what he calls "space debris," mainly for the station to appear good in photos. The engineering team tried but failed to convince the CEO that this was not needed. So instead they developed an innovative technology to maintain the surfaces while minimizing human operations outside the station. The maintenance is performed by several small robots moving over the space station surface, just like robotic vacuum cleaners. Before their first flight, Place-Y needs to assess the risks of collision during the operation of the robots. And this is exactly where you step in.

For the purposes of this problem, we model the space station as a collection of axis-aligned unit cubes (not necessarily connected). Each robot starts at time $t = 0$ in the center of an exposed face of one of the station's unit cubes (that is, a face which is not shared by a second station cube). The robot is oriented in one of the four directions parallel to an edge of the cube face. Every time unit, the robot moves straight ahead to another cube face, possibly pivoting 90 degrees across the space station edges so that it always maintains contact with the station. Note that if two cubes share an edge, the robot cannot slip between them (there is no gap).

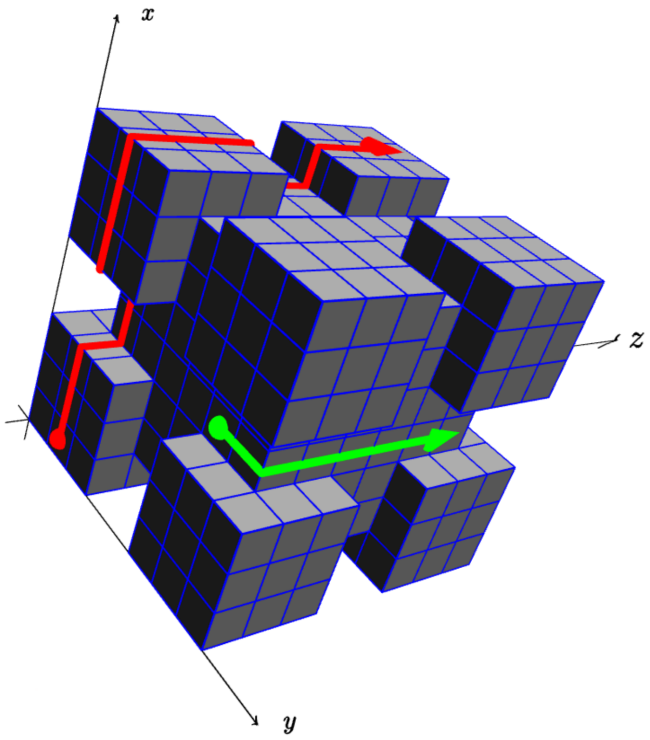


Figure K.1: Illustration of Sample Input 1. The dots denote starting points of two robots.

Given the layout of the station and starting positions of all the cleaning robots, determine the time of the earliest collision (if any). The time a collision occurs is either the time unit when two or more robots are on the interior of the same cube face or the time unit when two robots attempt to swap locations (see Sample Input 3 for the latter case).

Input

The first line of input contains two integers n and k, where n ($1 \le n \le 100$) is the number of regions describing the space station shape, and k ($0 \le k \le 100$) is the number of robots on the surface.

Each of the following n lines contains six integer coordinates x_1, y_1, z_1, x_2, y_2, and z_2 ($0 \le x_1 < x_2 \le 10^6$, $0 \le y_1 < y_2 \le 10^6$, $0 \le z_1 < z_2 \le 10^6$) describing one region and denoting that all the points x, y, z satisfying $x_1 \le x \le x_2$, $y_1 \le y \le y_2$, $z_1 \le z \le z_2$ are part of the space station. Note that some unit cubes may be included in more than one region.

Then follow k lines, each describing the starting position of one robot. Such a line contains three coordinates x, y, and z, and two directions $\vec{f}$ and $\vec{d}$. The coordinates specify that the robot starts at a face of the unit cube $(x, y, z) - (x + 1, y + 1, z + 1)$. The particular face is determined by $\vec{f}$ and the initial direction of movement is determined by $\vec{d}$. Both $\vec{f}$ and $\vec{d}$ are specified by one of the six strings `x+`, `x-`, `y+`, `y-`, `z+`, or `z-`, where `x+` designates the positive direction of the x-axis $(1, 0, 0)$, and so on. The axis letter in $\vec{f}$ will be different from the axis letter in $\vec{d}$. It is guaranteed that the starting cube belongs to the space station and the given face is an exposed face.

Output

Output the time of the first collision. If there will never be a collision, output `ok`.

Sample Input 1

```
9 2
1 1 1 7 7 7
0 0 0 3 3 3
5 0 0 8 3 3
0 5 0 3 8 3
0 0 5 3 3 8
5 5 0 8 8 3
5 0 5 8 3 8
0 5 5 3 8 8
5 5 5 8 8 8
0 1 0 z- x+
3 5 1 z- y+
```

Sample Output 1

```
44
```

Sample Input 2

```
1 3
0 0 0 1 1 1
0 0 0 x+ z+
0 0 0 y+ x+
0 0 0 z- y+
```

Sample Output 2

```
ok
```

Sample Input 3

```
1 2
0 0 0 2 1 1
0 0 0 y+ x+
1 0 0 y+ x-
```

Sample Output 3

```
0
```

Problem L
Sweep Stakes
Time limit: 20 seconds

You may have already won! In fact, you did already win! You won your very own island, in the deepest reaches of the unexplored ocean! Well, mostly unexplored. As it happens, there was a small military base there before you, and when they packed up and flew out they left behind an assortment of scraps, munitions, tunnels, ... and unexploded defensive ordnance. That's right: You now possess your very own minefield.

The minefield consists of an $m \times n$ grid, with any square of the grid holding 0 or 1 mines. Fortunately, you were able to recover the engineers' plans from when they deployed the mines. Unfortunately, the exact locations of the mines were never written down: the engineers had a preselected independent probability of deploying a mine in each square. However, you do know how many mines were placed in total.

You would like to estimate how safe various parts of your island are. Write a program to compute the probability of mine counts over various subsets of the minefield.

Input

The first line of input contains four integers m, n, t, and q, where m and n ($1 \le m, n \le 500$) are the dimensions of the minefield, t ($0 \le t \le mn$) is the total number of mines, and q ($0 \le q \le 500$) is the number of queries. The second line contains m real numbers $p_1, p_2, \ldots, p_m$ ($0 \le p_i \le 0.1$ for all i, with at most six digits after the decimal point specified), and the third line contains n real numbers $q_1, q_2, \ldots, q_n$ ($0 \le q_j \le 0.1$ for all j, with at most six digits after the decimal point specified). The preselected probability of the engineers placing a mine on square (i, j) is $p_i + q_j$. All choices of whether to place a mine on a given square were made independently, and the value of t is chosen so that the probability of deploying exactly t mines is at least 10^{-5}.

Each of the remaining q lines describes a single query. Each of those lines begins with an integer s ($0 \le s \le 500$), followed by s pairs of integers i and j ($1 \le i \le m$, $1 \le j \le n$), which are the coordinates of s distinct squares in the grid.

Output

For each query with s squares, output $s + 1$ real numbers, which are the probabilities of the s given squares containing $0, 1, \ldots, s$ mines. Your answer should have an absolute error of at most 10^{-6}.

Sample Input 1	Sample Output 1
2 2 1 2 0.05 0.05 0.05 0.05 1 1 1 2 2 1 1 2	0.75 0.25 0.5 0.5 0

Sample Input 2

```
3 4 3 4
0.02 0.04 0.06
0.005 0.07 0.035 0.09
1 3 2
3 1 4 2 4 3 4
4 1 2 2 3 3 1 1 4
8 1 1 1 2 1 3 2 1 2 3 3 1 3 2 3 3
```

Sample Output 2

```
0.649469772 0.350530228
0.219607636 0.527423751 0.237646792 0.015321822
0.267615440 0.516222318 0.201611812 0.014550429 0
0.054047935 0.364731941 0.461044157 0.120175967 0 0 0 0 0
```

Problem M
Trailing Digits
Time limit: 1 second

A large shipment of doodads has just arrived, and each doodad has a suggested retail price of b cents. You've noticed that consumers are much more likely to purchase goods when most of the trailing digits are the same. For example, items are more likely to be priced at 99 cents rather than 57 cents. So to make your goods more appealing, you've decided to sell your goods in bundles. To make a bundle, you choose a positive integer k, and sell k doodads for $k \times b$ cents. With an appropriate choice of k you can have a more pleasing price. For example, selling 57-cent doodads in bundles of size 7 means that each bundle sells for 399 cents, which has two trailing 9s, rather than no trailing 9s of 57. This idea of trailing 9s can be generalized to any other trailing digit: bundles of 692 57-cent doodads sell for 39 444 cents (three trailing 4s) and bundles of one million doodads sell for 57 000 000 cents (six trailing 0s).

After a little thought, you realize that you do not want to make your bundles too large—not only can the price be excessive, but who really needs several million doodads? For any type of doodad, your marketing department has a maximum bundle price of a.

Given the price of a doodad, the desired trailing digit, and the maximum price of a bundle, write a program that optimizes the trailing digits.

Input

Input consists of a single line containing three integers b, d, and a, where b ($1 \leq b < 10^6$) is the price of a doodad in cents, d ($0 \leq d \leq 9$) is the desired trailing digit, and a ($b \leq a < 10^{10\,000}$) is the maximum price of a bundle.

Output

Output the maximum number of consecutive occurrences of d that can appear at the end of a bundle price, given that the price of the bundle cannot exceed a.

Sample Input 1	Sample Output 1
57 9 1000	2

Sample Input 2	Sample Output 2
57 4 40000	3

Sample Input 3	Sample Output 3
57 4 39000	2

This page is intentionally left blank.

Problem N
What's Our Vector, Victor?
Time limit: 6 seconds

Image from pxfuel

Vector embeddings are a common tool in machine learning systems. Complex real-world concepts (for instance, words in a dictionary) are mapped onto vectors of real numbers. If embeddings are trained properly, related concepts remain close together in the vector space, so questions like "are these two words synonyms?" can be reduced to straightforward mathematical tests.

You have been hired by Ye Olde Ice Cream Shoppe to create an embedding model for flavors, so that when they are out of an ice cream flavor they can recommend related flavors to customers. After training your embedding on six cloud datacenters for four months, you finally had the perfect flavor model! You were ready to revolutionize the ice cream industry on your street and, dare we say it, your neighborhood! Well, until you accidentally dripped some free ice cream on your keyboard and deleted half the data. The Shoppe cannot afford the 30 billion rubles needed to retrain the model, so you are in trouble.

Fortunately, you still have various training results lying around. For a given deleted vector, the training data tells you how close it was to some known flavor vectors. The closeness of vectors A and B is just the standard Euclidean distance metric (that is, the length of the vector $A - B$). Write a tool that reconstructs embeddings which are consistent with the training results.

Input

The first line contains two integers d and n, where d ($1 \le d \le 500$) is the number of dimensions of the vectors, and n ($1 \le n \le 500$) is the number of training results for a deleted embedding vector you want to reconstruct. Each of the next n lines describes a training result using $d + 1$ numbers $x_1, \ldots, x_d$ and e. In a training result, $x_1, \ldots, x_d$ ($-100 \le x_i \le 100$) are the coordinates of a known vector, and e ($0 \le e \le 5\,000$) is the Euclidean distance from that vector to the deleted one.

Your submission will be tested only with the sample inputs given below and inputs generated according to the following procedure. First, d and n are chosen. Then, n input vectors and 1 output vector, each with dimension d, are chosen at random. The $d \cdot (n + 1)$ coordinates are independent and uniformly distributed in the interval $[-100, 100]$. Next, the Euclidean distance from each input vector to the output vector is computed. Finally, the output vector is discarded. Calculations use double-precision floating-point numbers. Numbers obtained using this procedure appear in the input with 17 digits after the decimal point.

Output

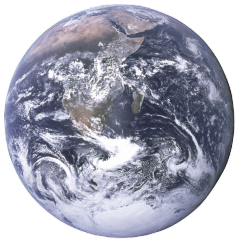
Output d values, the coordinates of any vector that has the given distance to each training vector with an absolute or relative error of at most 10^{-5}.

Sample Input 1	Sample Output 1
2 3 0 0 2.5 3 0 2.5 1.5 0.5 2.5	1.5 -2

Sample Input 2	Sample Output 2
2 2 0 0 2 4 -4 6	1.414213562373 1.414213562373

Sample Input 3	Sample Output 3
4 3 0 1 2 3 2 1 2 -1 7 5 1 0.3 3.4 1.2 3.3	1 2 3 4

Problem O
Which Planet is This?!
Time limit: 6 seconds



It's the year 2521, and interstellar probes have reached planets in distant solar systems. The Interstellar Consortium of Planet Cartographers (ICPC) has created detailed maps of these planets, and they seem to indicate the existence of alien life! On each map, the ICPC has recorded the locations of what appear to be alien dwellings.

The ICPC was planning to release this exciting news to the public, but at the last moment, disaster struck. One of the ICPC's interns deleted all meta-data associated with the maps. So while the maps themselves are safe, the ICPC does not know which maps belong to which planets. For this, they have come back in time to ask for your help. Given two maps, can you determine whether they describe the same planet? Hopefully, a 500-year head start will be enough time to solve this important problem!

The planetary maps consist of sets of points on the (spherical) planet surface. They are specified in terms of latitude (the angle north or south of the equator) and longitude (the angle west or east of the noon meridian, which is the location of the sun when the map's data was collected). Two maps for the same planet always agree on the latitudes of the points, since the planet's axis does not change. However, the longitudes of the points might differ, because the planet rotates between measurements.

Input

The first line of input contains an integer n ($1 \leq n \leq 400\,000$), the number of points in each of the two maps to be compared. Then follow n lines describing the first map. Each of these lines contains two real numbers a and b, where a ($-90 < a < 90$) is the latitude and b ($-180 < b \leq 180$) is the longitude. Coordinates are expressed in degrees and have at most four digits after the decimal point. No two points on the map will have the same coordinates. The remaining n lines describe the second map in the same format as the first.

Output

Output `Same` if there is a rotation around the planet's axis that transforms one map into the other. Otherwise, output `Different`.

Sample Input 1	Sample Output 1
4 0.0000 0.0000 30.0000 90.0000 -45.0000 -30.0000 30.0000 60.0000 30.0000 150.0000 30.0000 120.0000 0.0000 60.0000 -45.0000 30.0000	Same

Sample Input 2	Sample Output 2
3 0.0000 0.0000 30.0000 0.0000 30.0000 90.0000 0.0000 0.0000 30.0000 0.0000 30.0000 -90.0000	Different