A Very Long Bookshelf

task: `bookshelf`	points: 100

The village of Vyšná Boca is located in a long and narrow valley. During the winter, it is not unusual to get more than a meter of snow. On those days, the best thing you can do is to light the fireplace and climb into the bed with a good book.

Recently, the villagers decided that they are tired of reading the same books over and over again and they built a library. Since it had to fit into the valley, it was a long and narrow building with a single long bookshelf inside. From time to time a villager would come, select a contiguous sequence of books, and take them home. And from time to time a villager would come with a new set of books. He would then pick a place on the shelf, push the original books to the left and to the right to make enough space, and he would place the books he just brought into the gap.

Task specification

For the purpose of this task, the books will be identified by positive integers not exceeding $2 \cdot 10^9$. There can be multiple copies of the same book in the library. The best books have numbers smaller or equal than 100. The positions for the books are numbered from left to right, starting with zero. Initially, the bookshelf is empty.

Your task is to write a module that will simulate the actions in the library. Your module must provide the implementation of the following five functions:

- `void init()`
  `procedure init`
  This function will be called just once at the beginning.
- `int examine_book(int p)`
  `function examine_book(p : longint) : longint`
  This function shall return the book that is currently at position `p`. (The book remains on the bookshelf.)
- `void add_books(int k, int n, int *b)`
  `procedure add_books(k, n : longint; b : array of longint)`
  This function shall take the `n` books provided in the array `b` and insert them into the bookshelf starting at position `k`.
- `void remove_books(int k, int n)`
  `procedure remove_books(k, n : longint)`
  This function shall take `n` books starting with the book at position `k`, and remove these books from the bookshelf.
- `int best_books(int begin, int end)`
  `function best_books(begin,end : longint) : longint`
  This function shall return the number of the best books that are in range $begin, begin + 1, \ldots, end$. (The books remain on the bookshelf.)

Evaluation

Your module will be evaluated on many different test cases. A test case is considered solved if all the calls of your functions terminate correctly within the time limit, and all the calls to `examine_book` and `best_books` return the correct books or the correct number.

Each test case can be described by three numbers: the total number C of function calls our grader will make, the maximum number B of books at the bookshelf at any single time, the total number N of added or removed books, and the sum Q – size of all queries about the best books. Additionally, some test cases do not contain any calls to `remove_books`.

The limits for various test cases look as follows:

points	10	10	10	10	10	10	10	10	10	10
C	1000	2000	250000	250000	250000	500000	700000	500000	500000	500000
B	1500	2500	125000	125000	250000	700000	750000	750000	800000	10^6
N	1500	4000	250000	125000	250000	700000	750000	750000	$5 \cdot 10^6$	$5 \cdot 10^6$
Q	10000	10000	500000	10^9	10^6	10^{10}	10^7	10^7	10^{10}	10^9
remove?	no	yes	yes	no	no	no	no	no	yes	yes

Example

input

```
init()
add_books(0, 5, [1, 2, 3, 4, 5])
add_books(2, 2, [1000, 1002])
examine_book(3)
remove_books(3, 3)
best_books(1, 3)
```

output

```
examine_book(3): 1002
best_books(1, 3): 2
```

Slovak Double Cross

task: `doublecross`	points: 100

One of the national symbols of Slovakia is the Slovak coat of arms, shown in the picture on the right. The most important feature on the coat of arms is the silver double cross. You can also find this symbol on some of the Slovak Euro coins, and, of course, in this task.

The double cross can easily be painted in a bitmap. Here are two examples:

```
..........
....1.....          ..1..
..11111...          .111.
....1.....          ..1..
111111111.          11111
....1.....          ..1..
....1.....
```

Formally, a *double cross pattern* consists of one vertical and two horizontal segments of '`1`'s. Additionally, all the following requirements have to be met:

- The horizontal segments must not be in adjacent rows.
- The vertical segment must begin strictly above the horizontal segments and end strictly below them.
- The vertical segment must divide each horizontal segment into two equal halves.
- The upper horizontal segment must be strictly shorter than the lower one.

Note that the bitmap on the right contains the smallest possible double cross pattern.

Task specification

You are given a matrix of '`0`'s and '`1`'s. Output the value (D modulo $10^9 + 9$), where D is the number of occurrences of a double cross in the matrix.

As long as the conditions for the double cross pattern are satisfied, we do not care about the contents of the surrounding cells. (In the above examples, the positions with dots can contain both '`0`'s and '`1`'s.)

If the same pattern occurs in multiple locations, or if there are multiple overlapping occurrences of the double cross, all of these occurrences should be counted.

Input specification

The first line of the input contains two integers R and C: the number of rows and the number of columns of the matrix. Rows are numbered 1 through R from top to bottom, columns 1 through C from left to right.

The second line contains a single integer N – the number of cells in the matrix that contain '`0`'s.

N lines follow. Each of these lines contains two integers r_i and c_i – the coordinates (row and column) of one cell that contains a '`0`'. No two of these N cells are equal.

Constraints

The test cases are divided into several batches. For each of the batches, we give you the maximum values of R, C, and N in the test cases that form the batch, and the number of points awarded for solving it.

points	10	5	5	15	15	20	5	5	5	15
R	10	10	100	100	2500	4000	400	400	400	100
C	10	10	100	6000	50	30	3000	3000	3000	10000
N	100	5	15	1000	200	1000	1	5	15	10000

Additionally, the first batch is very easy. In this batch, each test case has the property that all '`1`'s in the matrix form a double cross pattern.

Output specification

Output a single line containing the value (D modulo $10^9 + 9$), where D is the number of ways in which a double cross pattern can be located in the matrix.

Example

input

```
6 8
12
1 2
1 3
1 4
1 6
2 2
3 2
3 3
3 4
3 7
6 4
6 6
4 8
```

output

```
5
```

This is the matrix described by the input:

```
10001011
10111111
10001101
11111110
11111111
11101011
```

All five valid double cross patterns are shown below.

```
....1...   ....1...   ....1...   ....1...   ....1...
...111..   ...111..   ...111..   ...111..   ..11111.
....1...   ....1...   ....1...   ....1...   ....1...
..11111.   ..11111.   ....1...   ....1...   ....1...
....1...   ....1...   ..11111.   .1111111   .1111111
........   ....1...   ....1...   ....1...   ....1...
```

The sequence jumps up and down

task: `updown`	points: `100`

The *characteristics* of an n-element integer sequence $a_1, \ldots, a_n$ is a string S of length $n-1$, where:

$$S[i] = \begin{cases} \texttt{I} & \leftarrow a_i < a_{i+1} \text{ (sequence increases)} \\ \texttt{C} & \leftarrow a_i = a_{i+1} \text{ (sequence is constant)} \\ \texttt{D} & \leftarrow a_i > a_{i+1} \text{ (sequence decreases)} \end{cases}$$

For example, the characteristics of $(3, 1, 1, 5, 6, 2)$ is `DCIID`.

Some sequences are boring. For example, the sequence $(1, 5, 2, 7, 3, 6, 2, 8)$ is pretty boring, because it just jumps up and down. This can be nicely seen from its characteristics: `IDIDIDI`.

We will now formally define which sequences are boring:
For a given sequence, its *level of boringness (LoB)* is the largest integer k such that its characteristics contains two equal (possibly overlapping) substrings of length k.

For example, the sequence $(1, 5, 2, 7, 3, 6, 2, 8)$ has LoB=5.

Task specification

Given k, produce the *longest possible* integer sequence with LoB strictly less than k.
Additionally, your sequence must use the *fewest possible* number of distinct values.
Partial credit will be awarded for sequences that only meet the first requirement.
The source code size for this problem must not exceed 10 000 characters.

Input specification

The only line of the input file contains the integer k ($1 \leq k \leq 14$).
All possible values of k will be used as test cases. Larger k are worth more points.

Output specification

Output your sequence, one element per row. Each element must fit into a signed 32-bit integer variable.

Example

input	output
2	1 2 3 3 3 2 47 2 1

The characteristics of this sequence is `IICCDIDD`.
Clearly, no substring of length 2 is repeated, so this sequence has LoB=1. However, this is not the longest such sequence – this output would not receive any points.

Boringness of books

task: `boringness`	points: 100

Now you know what boringness of sequence exactly means. On closer look we find out that the boringness of many other things could be measured in this way – books, movies, music etc.

For example, the level of boringness of a book is the largest integer k such that its text contains two equal substrings of length k. This definition is almost perfect, because nothing is so boring as reading the same text twice.

Many libraries (including the one in Vyšná Boca) have decided to enhance their catalogues by the level of boringnes of each book. Unfortunately, they discovered that it is quite hard to calculate the level quickly. So they asks you for help.

Tasks specification

Your task is to write a program, which calulates the level of boringness of the given text as fast as possible. You may assume, that the input text consists only of small letters of english alphabet.

Input specification

The first and only line of input contains the whole input text. The lenght of the text is always lower than 2^{20}.

Output specification

The only line of output contains one non-negative integer k, which is equal to the level of boringness of the input text. It is the lenght of two longest equal substrings in the text which start on different positions. These substrings may possibly overlap.

Example

input	output
`mississippi`	`4`

The substring 'issi' is contained twice in the string 'mississippi' and is the longest one. Its lenght is 4 letters. There exist no other same substrings which would be longer.

Revenge

task: `revenge`	points: 100

Image that you met your dreamgirl and she asked for help with her programming class homework – a program for searching the minimal spanning tree of the given graph. You shyly whispered that you would be glad to help her. She gave you a big kiss and told you that you were the best guy all over the world. And later that day you saw her with another guy in restaurant...

Now you feel miserable and angry and you plan a terrible revenge. But you don't know which? It doesn't matter, we help you. Write her a program which instead of the smallest spanning tree finds the one which is strictly greater than the smallest one but as small as possible. She surely won't find this bug in the program and your revenge will be finished.

Task specification

Write a program which for the given connected weighted graph finds the spanning tree which is strictly greater than the smallest spanning tree but as small as possible. By the spanning tree of the graph G, we mean such a subset of edges, so that graph G restricted only on these edges is still connected, but if we remove an arbitrary further edge, it splits into two parts. Notice, that it implies that spanning tree has always $N-1$ edges. The size of the spanning tree is the sum of sizes of its edges.

Input specification

On the first line of input, there are two integers N and (M $1 \leq N \leq 20\,000, 1 \leq M \leq 100\,000$) where N is the number of vertices and M is the number of edges in the graph G. In the following M lines of the input, there are always 3 numbers a, b and ℓ ($0 \leq a, b \leq N-1$, $0 \leq \ell \leq 100\,000$) which describe one edge from vertex a to vertex b of length ℓ. The graph G is always connected.

For 80% of the input data it holds that $N \leq 2\,000$.

Output specification

The only line of the output contains the size of the spanning tree which is strictly greater than the smallest spanning tree but as small as possible.

Example

input

```
9 13
1 7 3
2 7 2
7 8 5
7 5 2
6 7 1
5 6 3
4 6 2
3 4 7
3 6 5
7 3 4
8 6 4
0 2 1
0 3 8
```

output

```
20
```

Triangles

task: `triangles`	points: 100

Once there was a famous sailing ship contest called Sejly. Every year many experienced sailors competed in it. For this year the annual 100th Sejly is planned and its organizers want to have more competing sailors than ever. So they have decided to make billboards with a nice photo from the previous year to attract new sailors. Unfortunately this year the price of green color has extremely increased and unfortunately it is also the color of all the sails. So they need to count how much green color is needed for a single billboard and thence also how many billboards they can afford. Fortunately the rules of the Sejly are very restrictive – all the sails must be right-angled isosceles triangles. To make things easier you know that the coordinates of the vertices of all the sails (triangles) are integers. So they ask you – the young promising programmers – to help them with the problem. Your task is to compute the area of all the sails on the billboard.

Task specification

You are given a set of isosceles right-angled triangles with integer coordinates, having legs parallel to the coordinate axis and hypotenuses going "from upper left to lower right" and you are to output area of their union.

Input specification

On the first line there is a number n ($1 \leq n \leq 10\,000$) – the number of triangles. Following n lines will contain three space-separated integers x, y, d describing each triangle. A 3-tuple x, y, d corresponds to a triangle ABC with the following coordinates: $A = [x, y]$, $B = [x + d, y]$, $C = [x, y + d]$. You may assume that $0 \leq x, y, d \leq 1\,000\,000$.

Output specification

Output will consist of a single number u – the area of the union of given triangles. It should be printed with exactly one digit after the decimal point. You may assume that $u < 2^{31}$.

Example

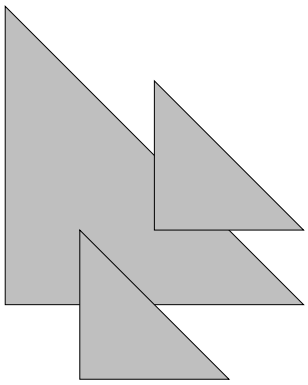
input

```
3
1 1 4
2 0 2
3 2 2
```

output

```
11.0
```

Corresponding picture is shown below.



Stack calculator

task: `calculator`	points: 100

Johnny has a weird calculator — a *stack* calculator. It consists of K memory cells (initially empty) numbered from 1 to K and it supports the following operations:

„`1`“ Writing the number 1 to the free cell with the lowest index. If there is no such cell, an exception is thrown.

„`+`“ Reading two numbers from occupied memory cells with the highest indices, freeing these two cells and writing their sum in the free cell with the lowest index. If there are less than two occupied memory cells, an exception is thrown.

„`-`“ Reading two numbers from occupied memory cells with the highest indices, freeing these two cells and writing their difference (the cell with the second highest index minus the cell with the highest index) in the free cell with the lowest index. If there are less than two occupied memory cells or if the result of this operation is a negative number, an exception is thrown.

„`*`“ Reading two numbers from occupied memory cells with the highest indices, freeing these two cells and writing their product in the free cell with the lowest index. If there are less than two occupied memory cells, an exception is thrown.

The calculator display always shows the number from the occupied cell with the highest number. If there is no such cell, the display is empty.

Every operation takes precisely one second. Starting with an empty calculator, what is the minimum number of seconds required to get the number N shown on the display?

Input

The first line of the input contains two natural numbers: N and K ($1 \leq N \leq 10^9$, $2 \leq K \leq 100$).

Output

Print one integer — the minimal time required to get the number N shown on the display.

Example

input	output
`6 3`	`9`

One optimal solution: `111++11+*`.

input	output
`11 4`	`15`

One optimal solution: `11+111++*11+*1-`.

Graffiti on the fence

task: `fence`	points: 100

In Byteland there is a long fence, consisting of N planks numbered from 1 to N. The Byteland government wants to have the fence painted. They hired M graffiti artists.

At the beginning, each artist is standing at one of the planks. Each artist can paint any plank he's standing at in b seconds, and walk to an adjacent plank in a seconds. An artist can also do nothing and just wait for inspiration.

The government has some requirements. They want to buy as little paint as possible, so they want each plank to be painted exactly once. They also want the entire fence done as soon as possible.

Task description

Given the length of the fence, starting positions of the artists and the values a and b, calculate the minimal possible time in which the artists can complete their work.

Input

In the first line there are two integers: N and M ($1 \leq N, M \leq 100\,000$). In the next line there are two integers: a and b ($1 \leq a, b \leq 10^6$). In the last line there are M integers $p_1, p_2, \ldots, p_M$ denoting the initial positions of the graffiti artists ($1 \leq p_i \leq N$).

Additionally, in test cases worth 40% of points we have $M \leq 2$.

Output

Print one integer – the minimal time required to paint the entire fence.

Example

input

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

output

```
5
```

It is possible to paint the fence in 5 minutes if artist number 1 paints plank number 2, artist number 2 paints plank number 1 and artist number 3 paints plank number 3. In this example, the fourth artist does nothing.

Permian garden

task: `garden`	points: 100

The orangery "Permian garden" is a rectangular building in which Permian plants grow. The orangery used to be divided into squares with paths. In the center of every square (and only there) there would grow exactly one plant. The size of the square was dependent on the root system of the plant.

Over a year the paths were overgrown with grass, which made moving around the orangery considerably more difficult. In order not to damage the roots of any plant during the gardening work, the size of its corresponding square should be determined based on its location.

We shall introduce the Cartesian coordinate system, which beginning we shall bind to the left lower corner of the orangery. The OX axis will coincide with the lower side of the building and the OY axis will coincide with the left side. Originally the paths were parallel to the coordinate axes. The unit segment was picked so that the corners of the squares have integer coordinates.

Task statement

Write a program that, given the size of the orangery and coordinates of the plants, will recover the squares corresponding to the plants.

Input

In the first line of the input there are integers W – the width of the orangery, H – the height of the orangery and N – the number of plants inside. ($1 \leq W, H \leq 10^{12}$, $1 \leq N \leq 2 \cdot 10^5$) In the next N lines given are the coordinates of the plants – two numbers x_i, y_i ($0 < x_i < W$, $0 < y_i < H$).

Output

Print N integers – the lengths of the sides of the squares corresponding to consecutive plants from the input.

Example

input

```
4 6 3
1 1
3 1
2 4
```

output

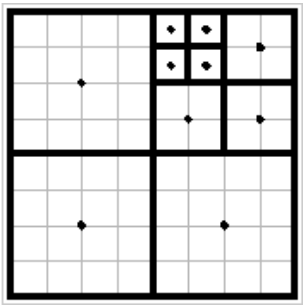
```
2
2
4
```

input

```
8 8 10
4.5 7.5
5.5 7.5
2 6
4.5 6.5
7 7
5 5
6 2
7 5
2 2
5.5 6.5
```

output

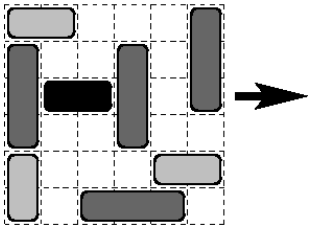
```
1
1
4
1
2
2
4
2
4
1
```



The figure on the right shows the correct output for the second example.

Traffic jam

task: `cars`	points: 100



Traffic jam is a real nightmare of all drivers. Nobody likes to be stuck in the overfilled streets, when the cars move very slowly, if they even move at all. Professional drivers face traffic jams quite often. Can you help them to find the way out of the traffic jam?

We can model a small (but complicated) traffic jam on a 6×6 grid of squares. Vehicles (cars and trucks) are scattered over the grid at integer locations, as shown below. Both types of vehicles are 1 square wide. Cars are 2 squares long, and trucks are 3 squares long. Vehicles may be oriented either horizontally (East-West) or vertically (North-South) relative to the grid.

Vehicles cannot move through each other, cannot turn, and cannot move over the edge of the grid. They can move in their direction (horizontally-oriented vehicles cannot move vertically and vice versa), as long as they are not blocked by another vehicle or by the edge of the grid. Only one vehicle may move in a single step, but it may move by as many squares at a time as possible, providing there is enough empty space.

Our goal is to move vehicles back and forth until a particular horizontally-oriented vehicle (your own car – the black one on the picture above) leaves the rightmost (eastern-most) edge of the grid, where it is considered to have escaped the traffic jam.

Task statement

You are to write a program that will find a solution requiring the minimum possible number of moves.

Input

On the first line there is a single integer n ($1 \leq n \leq 10$) giving the number of vehicles in the traffic jam. The input continues with n lines, each of them containing a description of one vehicle. The first character of the vehicle description is either `h` (meaning that the vehicle is oriented horizontally) or `v` (the vehicle is oriented vertically). It is followed by a space and two integers r, c ($1 \leq r, c \leq 6$) separated by a space. The integers specify the upper-left square occupied by the vehicle. Finally there is a space and either the character `c` or `t` determining whether the vehicle is a car or a truck. The first vehicle in the description is the one that should leave the grid and you can assume that it is a car and that it is oriented horizontally.

Output

The output should contain the minimal number of moves needed to move the first car out of grid over its right end, or the string `'The car is trapped.'` in case it is not possible to move the first car out.

Examples

input

```
8
h 3 2 c
h 1 1 c
h 5 5 c
h 6 3 t
v 2 1 t
v 5 1 c
v 2 4 t
v 1 6 t
```

output

```
8
```

The input corresponds to the picture above.

input

```
3
h 1 1 c
v 1 6 t
v 4 6 t
```

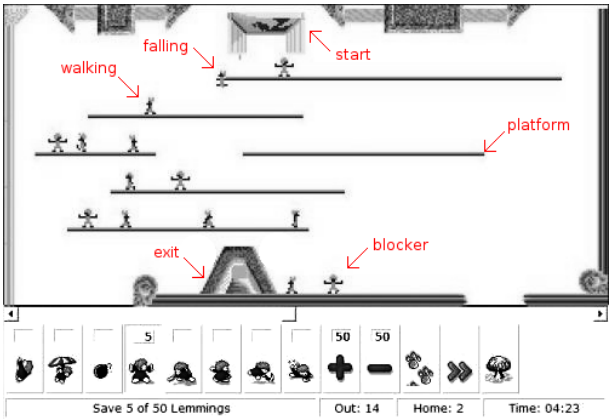
output

```
The car is trapped.
```

Lemmings

task: `lemmings`	points: 100

Lemmings is a very famous game. In this game, your task is to guide a group of small suicidal creatures with green hair. For this purpose, you may force some of them to do a specific task like digging, blocking the way, building the stairs, and several more. Writing a program to solve Lemmings is (provably!) a very hard problem – it is very unlikely that a polynomial time solution exists. In this task you have to solve a much simpler version of this game – the only two "skills" available will be the blocker and the nuke. (This is explained below in more detail.)



At the start of the game (time $t = 0$), the starting trapdoor opens and lemmings start falling out in 1 second intervals, for a total of L lemmings. All lemmings are initially facing right. Your task is to guide as many lemmings as possible to the exit. The game ends when there are no living lemmings inside the game (i.e., all lemmings are either dead or reached the exit).

Each level of our simplified lemmings game consists of:

- The starting trapdoor.
- Several horizontal platforms.
- An infinitely wide water at height $y = 0$.
- The exit.

For the purpose of our problem, the start, the exit and all lemmings are points. The exit is considered reached as soon as a lemming comes within a very small distance ε of it.

A lemming stands on a platform iff it is located at (x, y) such that for some platform p we have $y = p_y$ and $x \in [p_{start}, p_{end}]$. An alive and mobile lemming is either *walking* or *falling* at any moment – if it is standing on a platform, it is walking, otherwise it is falling. Both movement types have a constant speed of 1 unit per second.

Any walking lemming can be instantly converted into a *blocker*. This lemming remains standing at its current location forever. Whenever another lemming reaches this location, its walking direction is reversed.

There are three ways for a lemming to die:

- If the lemming falls onto a platform from a distance larger than H, it is immediately smashed by the impact.
- If the lemming falls into the water, it will immediately drown. Note that there can be a platform at $y = 0$, in which case the lemming only drowns when dropping down from its end.
- At any moment during the game the user may nuke the level. This initiates a countdown sequence of 10 seconds. After the countdown terminates, all lemmings that did not reach the exit explode and die. (Including lemmings that did not enter the level yet.) If the lemming reaches exit in the exact time of detonation, it will be saved.

  Note that nuking is necessary whenever you used at least one blocker, because without nuking the game would never terminate.

Your task is to find the maximal number of lemmings that can be safely guided to the exit and the shortest time needed to do so.

Input

The first line of input contains two integers L, H ($1 \leq L \leq 10^9$, $1 \leq H \leq 10^9$) – the number of lemmings on the start and the maximal safe falling distance. The second and third line both contains two space-separated integers x ($-10^9 \leq x \leq 10^9$) and y ($0 \leq y \leq 10^9$) – the position of the start and the exit.

The fourth line contains an integer N ($1 \leq N \leq 100\,000$) – the number of platforms. The next N lines will describe platforms: the $i+4$-th line contains three integers $xstart_i$, $xend_i$, y_i ($-10^9 \leq xstart_i < xend_i < 10^9$, $0 \leq y_i \leq 10^9$).

You may assume that

- no two platforms share a common point
- the start is not on any platform
- the lemming falling from the start does not land on a platform edge
- the exit is on one of the platforms
- the exit is not on a platform edge

Output

Output one line with two integers L' and T – the number of lemmings that can be saved and the time needed to do so. You should output integer T if the optimal time needed to end the level is within the interval $(T-\varepsilon, T+\varepsilon)$ for an arbitrarily small positive ε. If there are more possible solutions, print such that L' if maximal. If there are still more possibilities, minimize T.

Examples

input

```
1 1000
10 10
25 6
3
0 20 8
0 20 7
20 30 6
```

output

```
1 19
```

The only lemming will fall for 2 seconds, then walk to the right for $10+\varepsilon$ seconds. After this, it will start falling, miss the second platform and land safely on the third platform.
Ignoring the εs, the required time to complete the level is $2+10+2+5=19$ seconds.

input

```
3 1
10 9
35 5
4
0 20 8
0 20 6
20 30 7
20 40 5
```

output

```
1 31
```

Without blockers, the lemmings would fall from the point $(30+\varepsilon, 7)$ and die. To save them, the player needs to use two blockers. It is optimal to use them at $(20+\varepsilon, 7)$ and $(20-\varepsilon, 6)$. The third lemming will then safely reach the exit.

Post

task: `post`	points: `100`

There are plans to improve the mail delivery system in Byteland; Instead of lots of old mail trucks a single E-mobil will be introduced, which will be responsible for delivering mail to all post offices in Byteland;

The IT department of the city post has planned a cyclical route going through all post offices in the town. All connections on it are unidirectional. A post office which will be the center of the system is yet to be picked — all Byteland mail will go there before being delivered by the E-mobil. Because of traffic jams movement speed on the roads between consecutive offices on E-mobil's path depends on the current time. The center placement is optimal if after the E-mobil's departure at time zero it will deliver all the mail and come back to the center in the lowest possible time. You may assume that the time needed for unpacking mail is negligibly small.

Task statement

You have to write a program that, given the E-mobil's path and the schedule of speed limits will choose the optimal mail delivery center placement and the minimum time in which E-mobil can return.

Input

In the first line of the input there is a natural number N ($1 \leq N \leq 100\,000$)— the number of post offices in Byteland. The offices are numbered according to their order on the E-mobil's path, starting with 1. The next N blocks of lines describe the N consecutive road segments between the offices. Each road segment is described by three lines:

- In the first line there is a natural number d_i — the length of the given road segment ($1 \leq d_i \leq 10^9$), and a nonnegative integer E_i — the number of time intervals in which speed on this segment is constant.
- In the second line there are positive integers $t_{i,j}$ ($1 \leq j < E_i$, $0 < t_{i,j} \leq 10^9$) — the time moments in which the speed on the given road segment changes.
- In the third line there are positive integers v_j — the speeds during the time intervals $[t_{i,j-1}, t_{i,j})$ where $1 \leq j \leq E_i$. We assume that $t_{i,0} = 0$ and $t_{i,E_i} = \infty$.

You may assume that $\sum E_i \leq 100\,000$.

Output

In the only line of the standard output you should print the index of the post office which should become the mail delivery center and the minimum time for the E-mobil to make a full trip. The answer should have relative or absolute precision of at least 10^{-5}.

Examples

input

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

output

```
2 2.833333
```

input

```
2
2 1

2
2 1

2
```

output

```
1 2.000000
```