Problem A. Mountain Skyline

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

Sometimes a high mountain can look smaller than a lower one because it is further away. The low mountain could even hide the higher one entirely. Sometimes this makes it tricky to identify which mountain is which. Your task in this problem is to identify the mountains visible to an observer.

For this problem, assume a flat world. Each mountain is a downward-sloping cone with a slope of 45 degrees, so the radius of the base of each mountain (at altitude 0) is equal to its height. The cones of different mountains can intersect each other. The peak of each mountain is given as a triple of integer coordinates in a 3-dimensional Cartesian coordinate system, with $-10000 \le x, y \le 10000$, $0 < z \le 10000$. Here, z is the altitude of the peak mountain. The observer is at coordinate location 0, 0, 0. Neither the observer nor the peak of any mountain is inside (or on the border of) the cone of any other mountain. If a peak appears to the observer precisely behind the edge or peak of another mountain, it is considered **not** to be visible.

Input

The first line contains an integer $1 \le n \le 1000$, the number of mountains. The next n lines each contain three space-separated integers $-10000 \le x, y \le 10000$, $1 \le z \le 10000$, the coordinates of the peak of each mountain, followed by a sequence of at most 30 uppercase letters, the name of the mountain. All mountain names are unique.

Output

Output the names of the mountain peaks visible to the observer, one per line, in clockwise order. That is, start in the direction of the positive y-axis, then move towards to positive x-axis, then to the negative y-axis, then to the negative x-axis, and finally back to the positive y-axis. A mountain peak exactly on the positive y-axis should be listed at the beginning rather than at the end of the list. If two peaks appear one exactly above the other, list the higher one first.

Examples

standard input	standard output
3 0 10000 8849 EVEREST 10000 0 5959 LOGAN 0 -10000 4808 BLANC	EVEREST LOGAN BLANC
6 8 0 5 FUJI 9 1 5 MATTERHORN 9 0 5 KEBNEKAISE 9 -1 5 FAGRADALSFJALL 16 0 10 KILIMANJARO 120 0 80 DENALI	MATTERHORN DENALI FUJI FAGRADALSFJALL

Problem B. Noise

Input file:	`standard input`
Output file:	`standard output`
Time limit:	6 seconds
Memory limit:	1024 megabytes

You want to write an app where you can record a short fragment of a song and it will tell you which song it is. In your app you represent audio as an array of the amplitude of the audio wave every 100 microseconds (the "pulse-code modulation PCM" raw audio format).

For this problem, you are focusing on the case where we have a single song represented by the array $a_1, a_2, \dots, a_n$, and a recorded fragment represented by the array $b_1, b_2, \dots, b_m$. You now wish to count how many times the recorded fragment b occurs inside the song a.

However, you want to take into account that the microphone used to record b is not always perfect — there might be some noise added. In particular, if your microphone measured value b_i, it could be the case that the true value is actually $b_i - 1$, b_i or $b_i + 1$. Note that it is possible for the recording to have different noise in different entries. E.g. if the true array is $[1, 2, 3, 4]$ it could be the case that the recorded array would be $[2, 2, 2, 5]$.

You thus want to count the number of possible occurrences of b inside a, when taking this noise into account. In particular, given arrays a and b of length n and m respectively, you want to count how many offsets x ($0 \leq x \leq n - m$) there are such that $b_i - 1 \leq a_{x+i} \leq b_i + 1$ for all $1 \leq i \leq m$.

Input

- The first line contains two space-separated integers n and m ($1 \leq m \leq n \leq 400\,000$).
- The next line contains n integers: $a_1\ a_2\ \dots\ a_n$ ($1 \leq a_i \leq 1\,000\,000$).
- The final line contains m integers: $b_1\ b_2\ \dots\ b_m$ ($1 \leq b_i \leq 1\,000\,000$).

Output

Output a single integer, the answer to the problem.

Examples

standard input	standard output

5 3
1 2 3 4 5
2 3 4
	3

5 2
100 199 300 201 299
200 300
	2

3 3
1 1 1
1 2 3
	0

Problem C. Bus Connections

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

Sometimes there is a bus that goes directly from your current location to your destination, which is convenient. Other times, you may need to take two or more buses and switch between them to reach your final destination. It's annoying that when switching from one bus to another, you often need to wait for the next bus to arrive. Wouldn't it be nice if you could time your trip such that you never have to wait for a connection, that you reach a stop in one bus at exactly the time that your next connecting bus arrives?

Input

The first line of input contains an integer $1 \le b \le 1000$, the number of buses you need to take to reach your destination. The following b lines each describe the schedule of a bus route, in the order that you need to take them. Each line contains three space-separated integers $0 \le t \le 1,000,000$, $1 \le i \le 1000$, $1 \le d \le 1000$. The first integer t is the time at which the first bus on the route arrives at the stop at which you need to get on it. The arrival time is specified in minutes since noon on January 1, 2021. The second integer i specifies the interval of buses on this route. Thus, buses on this route will arrive at the stop at times t, $t+i$, $t+2i$, $t+3i$, ... The third integer d is the duration of time in minutes that you need to ride on this bus before reaching the stop at which you get off. After this time, you get off the current bus and get on the following one.

Output

Output a line containing a single integer, the earliest time (in minutes since noon on January 1, 2021) at which you should get on the first bus such that you can then transfer to all the other buses in sequence without ever having to wait when switching buses. If there is no such time, output `-1`.

Examples

standard input	standard output

3
101 5 100
100000 7 200
0 4 300
	99956

2
0 6 9
0 10 9
	-1

8
122 997 491
808 991 290
172 983 560
928 977 101
570 971 592
357 967 123
429 953 835
199 947 295
	5261736655538653840278l8

Note

First sample: Take the first bus at time 99956 (which departs every 5 minutes starting at time 101). Then you arrive at the second bus at time 100056, which perfectly times a departure. You finally swap from the second to third bus at time 100256.

Second sample: The arrival time of the first bus never coincides with the departure time of the second bus.

Problem D. Not Long Enough

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1.5 seconds
Memory limit:	256 megabytes

Alice has n 2-dimensional vectors but Bob thinks that these vectors are not long enough.

Alice wants to find a subset of these vectors such that their sum is as long as possible.

Input

First line contains a single integer n. Next n lines contain 2 integers each, x_i and y_i, coordinates of the i-th vector.

$1 \le n \le 2 \cdot 10^5$,

$-10^4 \le x_i, y_i \le 10^4$.

Output

Print one integer — squared length of the longest possible vector that can be created.

Example

standard input	standard output

4
1 0
0 1
1 1
-1 -1
	8

Note

In the sample, the sum of the first 3 vectors is (2, 2), resulting in the squared length of 8.

Problem E. Frogger

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

You may have played the classic video game Frogger. In this game, a frog must cross a busy street without getting run over by one of the many fast cars. Then the frog must cross a river by jumping onto floating logs without falling in the water.

In this problem, we focus on the second part of the game, in which the frog crosses a river using floating logs. The game is played on a rectangular grid. Each log is the size of one square of the grid. Unlike in the real game, logs can float vertically in addition to horizontally.

This means that logs can also collide with each other. In this case, multiple logs can just be on the same grid square at the same time, and they just pass through each other without changing their direction and speed. When this happens, this is the only time that the frog can switch from one log to another. At any time, the frog is always on some log and travels with that log. If at any point in time, the log that the frog is on is on the same grid square as another log, the frog may choose to switch and continue travelling on the other log.

The input describes the state of the river at time $t = 0$. Specifically, the input defines the position of each log and and the direction that it travels in, either up (`^`), down (`v`), left (`<`), or right (`>`). It is guaranteed that at time $t = 0$, there is some log in the top-left corner of the grid, where the frog starts. At each time step, all the logs move simultaneously, at the same speed, by one grid square in their specified directions. The grid wraps around: when a log reaches an edge of the grid, its next position is at the opposite edge of the grid.

Determine the earliest time that the frog reaches the bottom-right corner of the grid, or that it can never reach it.

Input

The first line contains two space-separated integers $2 \le R, C \le 50$, the number of rows and columns in the grid. The next R lines each contain C characters each, either a period (`.`) indicating no log on that grid square, or one of `^v<>` indicating a log and the direction in which it travels.

Output

Output one line containing the minimum number of time steps for the frog to reach the bottom-right corner of the grid, or `-1` if it can never reach the bottom-right corner.

Examples

standard input	standard output

`3 5`
`>^..v`
`...<.`
`.v..^`
	`5`

`3 5`
`<....`
`.v...`
`...>.`
	`31`

`2 2`
`><`
`><`
	`-1`

Note

In the first sample the frog can go right-down-left-left-down to reach the bottom-right corner.