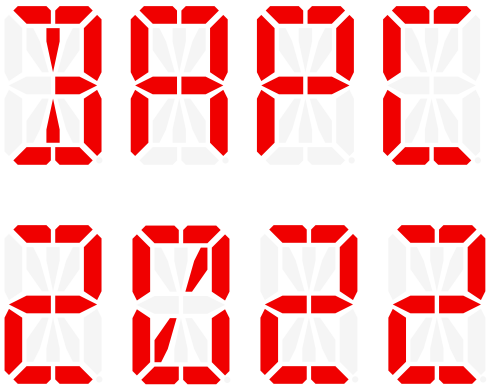
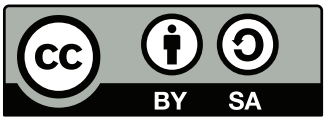
BAPC 2022

The 2022 Benelux Algorithm Programming Contest



Problems



A Adjusted Average

Time limit: 8s

As a student of the Biology And Probability Course at your university, you have just performed an experiment as part of the practical assignments. However, your results do not look very nice: you had hoped that the average of your samples would be different from what it is now.

"If you don't reveal some insights soon, I'm going to be forced to slice, dice, and drill!"

Torturing data. CC BY by Timo Elliott on timoelliott.com

To improve your results, you decide to let some of your samples "magically disappear" (i.e., dump them in the waste bin). In order to not raise suspicion with your teacher, you can remove only a few of your samples. How close can you possibly get to your desired average?

Input

The input consists of:

- One line with three integers n, k, and $\overline{x}$ ($2 \leq n \leq 1500$, $1 \leq k \leq 4$, $k < n$, $|\overline{x}| \leq 10^9$), the number of samples, the number of samples that may be removed, and the average you think looks the nicest.
- One line with n integers x ($|x| \leq 10^9$), representing the samples.

Output

Output the minimal absolute difference between $\overline{x}$ and the average you can obtain by removing at most k samples from the dataset.

Your answer should have an *absolute* error of at most 10^{-4}.

Sample Input 1	Sample Output 1
5 2 2 1 2 3 100 200	0

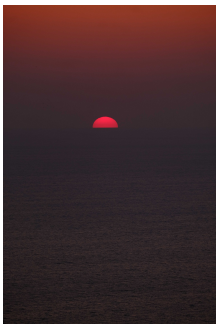
Sample Input 2	Sample Output 2
5 4 -5 -6 -3 0 6 3	0.5

Sample Input 3	Sample Output 3
4 1 4 1 3 3 7	0.333333333333333

B Bellevue

Time limit: 1s

As any photographer knows, any good sunset photo has the sun setting over the sea. In fact, the more sea that is visible in the photo, the prettier it is!



Sunset over the Mediterranean sea. CC0 by Ragnar Groot Koerkamp

You are currently visiting the island Bellevue, and you would like to take a photo of either the sunrise to the east or the sunset to the west to submit it to Bellevue's Astonishing Photography Competition. By carefully studying the topographic maps, you managed to find the east-west profile of the island. Now you would like to know the maximal amount of sea that you could capture in a photo, measured as the viewing angle covered by water.

The profile of the island is given as a piecewise linear function consisting of $n-1$ segments between n points. The island starts and ends at sea level. As an example, Figure B.1 shows the profile of the first sample case.

Note that the viewing angle of your lens is not large enough to capture the ocean to the east and west of the island in one shot. Also, the viewing angle of sea at sea level is 0 degrees.

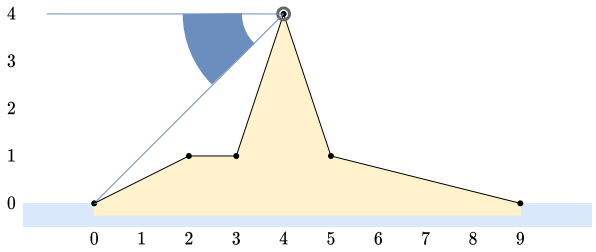


Figure B.1: The east-west profile of the island in the first sample case.

Input

The input consists of:

- One line with an integer n ($3 \leq n \leq 50\,000$), the number of points.
- n lines with two integers x_i and y_i ($0 \leq x_i, y_i \leq 50\,000$), a point in the east-west profile of the island.

It is guaranteed that the points are given from left to right ($x_1 < x_2 < \cdots < x_n$) and that the island starts and ends at sea level ($y_1 = y_n = 0$). The interior of the island is all above sea level ($y_i > 0$ for $1 < i < n$).

Output

Output the maximal viewing angle of sea you can see, in degrees.

Your answer should have an absolute or relative error of at most 10^{-6}.

Sample Input 1

```
6
0 0
2 1
3 1
4 4
5 1
9 0
```

Sample Output 1

```
45
```

Sample Input 2

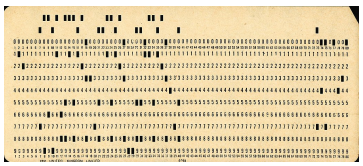
```
5
1 0
5 4
6 1
8 2
9 0
```

Sample Output 2

```
63.4349488
```

C Crashing Competition Computer

Time limit: 4s



A mid-twentieth century punchcard. CC BY 2.0 by Pete Birkinshaw on Wikipedia

You are writing code for a hackathon where you need to decode Binary ALGOL Punch Cards. You have already come up with the optimal solution and only need to type it out. The solution consists of c characters, and your typing speed is 1 character per time unit. However, your computer is prone to sudden crashes: after every character you type there is a probability of p that your computer crashes and you need to start over again. Recovering after a crash costs r time units, and you can then continue typing from the last point where you saved your code.

You can click "Save" at any time (which costs t time units) to save your code and to be able to restart from this point after crashes. Clicking "Save" will not cause your computer to crash.

Determine how many (expected) time units you need to complete the code. Note that the code should be saved after typing the last character.

Input

The input consists of:

- One line with three integers c, t, and r ($1 \leq c \leq 2000$, $0 \leq t, r \leq 10^9$), indicating the number of characters, the time cost of clicking "Save", and the time cost of recovering after a crash.
- One line with a floating-point number p ($0.001 \leq p \leq 0.999$, with at most 10 digits after the decimal point), the probability that your computer crashes after a character press.

Output

Output the expected number of time units you need to complete the code.

Your answer should have an absolute or relative error of at most 10^{-6}.

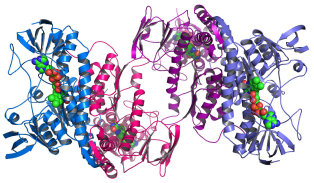
Sample Input 1	Sample Output 1
2 1 5 0.25	8.0

Sample Input 2	Sample Output 2
3 5 2 0.5	26.0

Sample Input 3	Sample Output 3
10 4 5 0.327	68.664967357

D Dividing DNA

Time limit: 2s



At the Bacteria And Protein Centre, you own a large collection of DNA. In fact, new strands of DNA come in all the time. To organise the vast amount of data, you identify each piece by its unique substrings: substrings that do not already occur in the database.

Your database can quickly determine whether a given piece of DNA occurs as a substring in the database or not. Naturally, if a certain DNA string is found in the database, it also contains all its substrings.

You now want to determine the *uniqueness* of a given piece of DNA: the maximal number of disjoint substrings it contains that are absent from the database.

You are given the length n of the query string $q_1 \dots q_n$, and you can repeatedly ask the database whether it contains the substring $q_i \dots q_{j-1}$.

As an example, consider the first sample interaction. In this case, the database contains strings "`TGC`" and "`CT`", and the query string is "`CTGCAA`". It has uniqueness 3, because it can be split into the new substrings "`CTGC`", "`A`", and "`A`". The new substring "`CTGC`" cannot be split up further: for example, the subdivision "`CT`" and "`GC`" is not allowed, because both substrings occur (possibly as substrings) in the database. Note that the actual characters in the string are not used in the interaction.

You may use at most $2n$ queries to the database.

Interaction

This is an interactive problem. Your submission will be run against an *interactor*, which reads from the standard output of your submission and writes to the standard input of your submission. This interaction needs to follow a specific protocol:

The interactor first sends one line with an integer n ($1 \leq n \leq 10\,000$), the length of the DNA piece for which you need to compute the uniqueness.

Then, your program should make at most $2n$ queries. Each query is made by printing one line of the form "`?` i j" ($0 \leq i < j \leq n$), indicating that you want to query whether the substring starting at position i and ending at position $j-1$ occurs in the database. The interactor will respond with either "`present`" or "`absent`", indicating whether the substring was found in the database.

When you have determined the uniqueness x of the piece of DNA, print one line of the form "`!` x", after which the interaction will stop. Printing the answer does not count as a query.

Make sure you flush the buffer after each write.

A testing tool is provided to help you develop your solution.

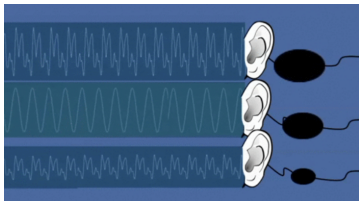
Using more than $2n$ queries will result in a wrong answer.

Read	Sample Interaction 1	Write
6		
	? 4 6	
absent		
	? 4 5	
absent		
	? 5 6	
absent		
	? 0 1	
present		
	? 0 2	
present		
	? 2 4	
present		
	? 1 4	
present		
	? 0 4	
absent		
	! 3	

Read	Sample Interaction 2	Write
10		
	? 0 10	
absent		
	? 0 9	
present		
	? 1 10	
present		
	! 1	

E Equalising Audio

Time limit: 4s



CC BY-SA 4.0 by Rburtonresearch on Wikipedia

As a radio engineer at the Balanced Audio Podcast © your job is to deliver an equal listening experience at all times. You did a poll among the listeners and they are especially concerned about fluctuations in loudness. To resolve this you bought a transformer to equalise the audio, but alas, its software got corrupted during transport.

Your job is to rewrite the equalising software. As input the transformer gets n amplitudes $a_1, \ldots, a_n$, with an average perceived loudness of $\frac{1}{n}\sum_{i=1}^{n} a_i^2$. The output should contain the same amplitudes, but renormalised by some constant positive factor, such that the average perceived loudness is x. There is one exception: total silence should always be preserved (i.e., when all amplitudes in the input are 0, they should remain 0).

Input

The input consists of:

- One line with a two integers n and x ($1 \leq n \leq 10^5$, $0 \leq x \leq 10^6$), the number of amplitudes and the average perceived loudness to achieve.
- One line with n integers $a_1, \ldots, a_n$ ($|a_i| \leq 10^6$), the amplitudes.

Output

Output one line containing n numbers, the renormalised amplitudes with an average perceived loudness of x.

Your answers should have an absolute or relative error of at most 10^{-6}.

Sample Input 1

```
5 6
0 1 -2 3 -4
```

Sample Output 1

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

Sample Input 2

```
4 1
1 3 3 7
```

Sample Output 2

```
0.242535625 0.7276068751 0.7276068751 1.697749375
```

F Failing Flagship

Time limit: 1s

Ahoy! You are sailing towards the next "Boats Are Pretty Cool" convention to sell your latest gadget: a new type of compass.

On a normal compass, it is difficult to read off the precise wind direction. However, your new type of compass lets you read off wind directions to a much higher precision! The display can display strings of at most 1000 characters.

Unfortunately, you have encountered some bad weather. After a few hours of heavy winds and big waves, you can finally look at your compass again. You read off the wind direction X you are going and know in which wind direction Y you need to go. However, to make the ship turn you have to enter the degrees of the angle the ship has to make in the control system. What is the smallest turn, in degrees, you have to make to get back on the right course?

The conversion of a wind direction to degrees goes as follows. The four basic wind directions are N, E, S, and W pointing at 0, 90, 180, and 270 degrees, respectively. There are also four wind directions consisting of two letters: NE, SE, SW, and NW, pointing at 45, 135, 225, and 315 degrees, respectively.

A wind direction can also consist of $k \geq 3$ letters $l_1 l_2 \ldots l_k$. In that case, the last two letters indicate one of the four two-letter wind directions, i.e., $l_{k-1} l_k \in \{\text{NE}, \text{SE}, \text{SW}, \text{NW}\}$ and the other letters are equal to one of these, i.e., $l_i \in \{l_{k-1}, l_k\}$ for all $i \leq k-2$. This wind direction points precisely in the middle of the following two wind directions:

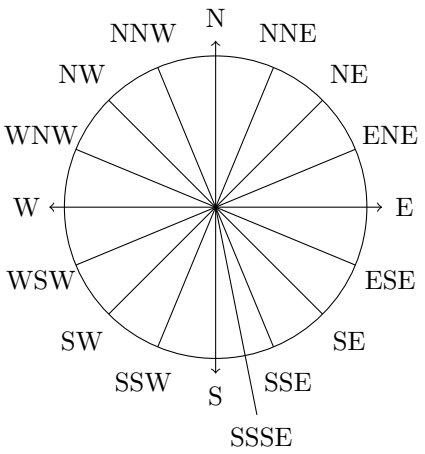


Figure F.1: Wind directions

- wind direction $l_2 \ldots l_k$,
- the first wind direction of at most $k-1$ letters you encounter when starting in $l_2 \ldots l_k$ and move along the circle towards l_1.

For example, the wind direction SSSE points in the middle of SSE and S, because S is the first wind direction with at most 3 letters when moving from SSE towards S, as can also been seen in Figure F.1.

Input

The input consists of:

- One line with two strings X and Y ($1 \leq |X|, |Y| \leq 1000$), indicating the wind directions as described above.

Output

Output the smallest angle of the turn you have to make to go from direction X to Y, in degrees. The angle is a non-negative number, irrespective of whether it describes a clockwise or counter-clockwise turn.

Your answer should have an *absolute* error of at most 10^{-6}.

Sample Input 1	**Sample Output 1**
`N S`	`180`

Sample Input 2	**Sample Output 2**
`NNE SSSE`	`146.25`

Sample Input 3	**Sample Output 3**
`ENE NW`	`112.5`

G Grinding Gravel

Time limit: 4s

Perfectly ground gravel in a perfect grid.
CC BY-NC 2.0 by markjowen66 on Flickr

During the renovation of your garden, you decide that you want a gravel path running from the street to your front door. Being a member of the Boulders And Pebbles Community, you want this path to look perfect. You already have a regular grid to put the gravel in, as well as a large container of gravel containing exactly as much as the total capacity of the grid.

There is one problem: the gravel does not yet fit perfectly into the grid. Each grid cell has the same (fixed) capacity and every piece of gravel has a certain weight. You have a grindstone that can be used to split the stones into multiple pieces, but doing so takes time, so you want to do a minimal number of splits such that the gravel can be exactly distributed over the grid.

As an example, consider the first sample case. There are three grid cells of size 8, which can be filled as follows. Put the stones of weight 2 and 6 in the first cell. Now grind the stone of weight 7 into two pieces of weight 3 and 4. Then the other two grid cells get filled by weights $3, 5$ and $4, 4$ respectively.

Input

The input consists of:

- One line with two integers n and k ($1 \leq n \leq 100$, $1 \leq k \leq 8$), the number of pieces of gravel and the capacity per grid cell.
- One line with n integers $w_1, \ldots, w_n$ ($1 \leq w_i \leq 10^6$ for all i), the weight of each piece of gravel.

It is guaranteed that $w_1 + w_2 + \cdots + w_n$ is a multiple of k.

Output

Output the minimal number of times a stone needs to be split into two, such that all the pieces of gravel can be used to fill all the grid cells perfectly.

Sample Input 1	Sample Output 1
5 8 2 4 5 6 7	1

Sample Input 2	Sample Output 2
2 5 12 13	4

H House Numbering

Time limit: 4s

Numbers. Pixabay License

You are addicted to the latest world-simulation game: Building A Perfect City. In your current play-through, you have created a city that has an equal number of streets and intersections. All that is left is to number the houses in every street.

The city is represented by a connected graph with intersections and streets. Every street is a connection between two intersections u and v, and has h houses which are all on one side of the street. There is at most one street between two intersections. There are two ways to number the houses in this street: either you start with house number 1 adjacent to intersection u and end with house number h at intersection v, or house number 1 is adjacent to v and house number h is adjacent to u. To avoid confusion, you want to ensure that no intersection has two adjacent houses with the same number.

Find a way to number the houses in every street that satisfies this property (or report that it is impossible).

Input

The input consists of:

- One line with an integer n ($3 \leq n \leq 10^5$), the number of intersections and number of streets.
- n lines with three integers u, v, and h ($u \neq v$, $1 \leq u, v \leq n$, $2 \leq h \leq 10^9$) representing a street between intersections u and v that has h houses.

It is guaranteed that every intersection is reachable from every other intersection. There is at most one street between any two intersections.

Output

If it is impossible, output "`impossible`". Otherwise, output for each street (in the same order as the input) a number representing the intersection where the house numbering starts.

If there are multiple valid solutions, you may output any one of them.

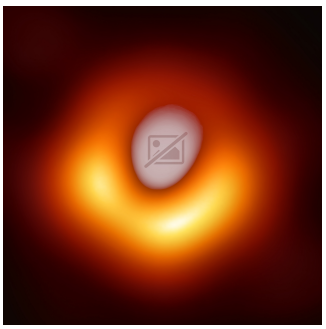
Sample Input 1	Sample Output 1
3 1 2 2 2 3 9 3 1 3	1 2 3

Sample Input 2	**Sample Output 2**
4 1 2 2 1 3 2 2 3 2 1 4 2	impossible

I Imperfect Imperial Units

Time limit: 4s

You are writing a paper for the Beta Astronomy Physics Conference about your recent discovery on grey holes. One of your collaborators has performed a huge number of measurements, which you would like to analyse in order to draw some conclusions. The only problem is: the data is measured in a wide variety of units, and to your disgust, they appear to use a mix of the imperial and metric systems. To simplify your analysis, you need to convert all these measurements into a different unit.



False-colour image of a grey hole. CC BY 4.0 European Southern Observatory, modified

Input

The input consists of:

- One line with two integers n and q ($1 \leq n \leq 100$, $1 \leq q \leq 10\,000$), the number of unit conversion equations and the number of queries to answer.
- n lines, each defining a unit conversion in the format "`1 <unit> = <value> <unit>`".
- q lines, each with a query in the format "`<value> <unit> to <unit>`".

In these formats, "`<value>`" is a floating-point number v ($0.001 \leq v \leq 1000$, with at most 9 digits after the decimal point) and "`<unit>`" is a string of at most 20 English lowercase letters (`a–z`). A unit in a query is guaranteed to be defined in at least one unit conversion equation. Every unit can be converted into every other unit in *at most* one way.

Output

For every query, output the value of the requested unit, or "`impossible`" if the query cannot be answered.

Your answers should have a *relative* error of at most 10^{-6}.

Sample Input 1

```
4 3
1 foot = 12 inch
1 yard = 3 foot
1 meter = 100 centimeter
1 centimeter = 10 millimeter
750 millimeter to meter
42 yard to inch
10 meter to foot
```

Sample Output 1

```
0.75
1512
impossible
```

Sample Input 2

```
4 3
1 fortnight = 14 day
1 microcentury = 0.036525 day
1 microcentury = 1000 nanocentury
1 week = 7 day
22.2 fortnight to nanocentury
2.5 nanocentury to week
3.14 day to fortnight
```

Sample Output 2

```
8509240.2464065708427
1.3044642857142857142e-05
0.22428571428571428572
```

Sample Input 3

```
10 2
1 micrometer = 1000 nanometer
1 millimeter = 1000 micrometer
1 meter = 1000 millimeter
1 kilometer = 1000 meter
1 megameter = 1000 kilometer
1 lightsecond = 299.792458 meter
1 lightminute = 60 lightsecond
1 lighthour = 60 lightminute
1 lightday = 24 lighthour
1 lightyear = 365.25 lightday
42 nanometer to lightyear
42 lightyear to nanometer
```

Sample Output 3

```
4.439403502903384947e-18
3.9735067984839359997e+20
```

J Jagged Skyline

Time limit: 4s

The future is here! The Boxes And Parcels Centre has decided to start delivering parcels using drones. Being a BrAinPort Company, naturally the first deliveries will be to Eindhoven.

The Eindhoven skyline.
CC BY 2.5 by Experience040 on Wikipedia

To keep the flight logic simple, the first prototype will only deliver to the roofs of the tallest buildings. After take-off, the drone will take a $w \times h$ ($1 \leq w \leq 10\,000$, $1 \leq h \leq 10^{18}$) photo of the skyline, as shown in Figure J.1. You have been tasked with the problem of determining the location and height of the tallest building in this photo, so that the drone knows where to go.

You have access to a classifier that can determine for each pixel whether it is "`sky`" or "`building`". You can use this multiple times for different pixels. To avoid unnecessary delays, you may run the classifier at most 12 000 times.

It is guaranteed that the buildings will not contain any hovering parts: whenever a pixel that is not on the bottom row of the photo is classified as building, the pixels below it will also be classified as building.

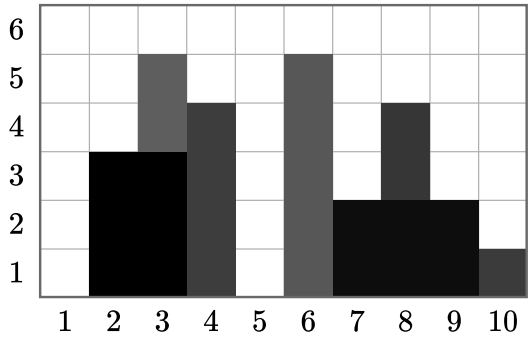


Figure J.1: The skyline of the sample interaction.

Interaction

This is an interactive problem. Your submission will be run against an *interactor*, which reads from the standard output of your submission and writes to the standard input of your submission. This interaction needs to follow a specific protocol:

The interactor first sends one line with two integers w and h ($1 \leq w \leq 10\,000$, $1 \leq h \leq 10^{18}$), the width and height of the photo in pixels.

Then, your program should make at most 12 000 queries to find the highest building. Each query is made by printing one line of the form "`?` x y" ($1 \leq x \leq w$, $1 \leq y \leq h$). The interactor will respond with either "`sky`" or "`building`", indicating the classification of the pixel at coordinates (x, y).

When you have determined the x-coordinate of one of the highest buildings and its height y, print one line of the form "`!` x y" ($1 \leq x \leq w$, $0 \leq y \leq h$), after which the interaction will stop. Printing the answer does not count as a query.

The interactor is not adaptive: the skyline is fixed up front, and does not depend on your queries.

If there are multiple valid solutions, you may output any one of them.

Make sure you flush the buffer after each write.

A testing tool is provided to help you develop your solution.

Using more than 12 000 queries will result in a wrong answer.

Sample Interaction 1

Read	Write
`10 6`	
	`? 1 1`
`sky`	
	`? 3 5`
`building`	
	`? 7 3`
`sky`	
	`? 9 2`
`building`	
	`! 3 5`

K Kiosk Construction

Time limit: 8s

You are planning to start a Beautifully Arranged Placid Camping. You already have bought a field, which you have divided into a $h \times w$-grid of plots, and numbered them with distinct numbers a_{ij} from 1 to $h \cdot w$. However, you forgot one thing: you still need to place the reception kiosk at one of the plots. You want to minimise the maximal distance that any guest will walk from the reception kiosk to their plot. Guests will however not take the shortest path to their plot, but instead they follow the following procedure, starting at the reception kiosk:

- Look at the numbers of the four neighbouring plots.
- Go to the plot with the number closest to the destination number. In case of a tie, out of the two tying plots, go to the one with the number closest to the current plot number.
- Repeat until the destination is reached.

Note that this procedure might not terminate in some cases.

Given the numbering of the plots, find the plot number of the optimal position for the reception kiosk and the maximal walking distance to any plot from this kiosk. If, for every possible position for the reception kiosk, there is at least one plot for which the procedure outlined above does not terminate, output that this is impossible.

Figure K.1 shows the third sample case: one solution is to put the kiosk in plot 4, so that every other plot is at most distance 3 away. Placing the kiosk in plot 7 does not work as plot 9 cannot be reached from there.

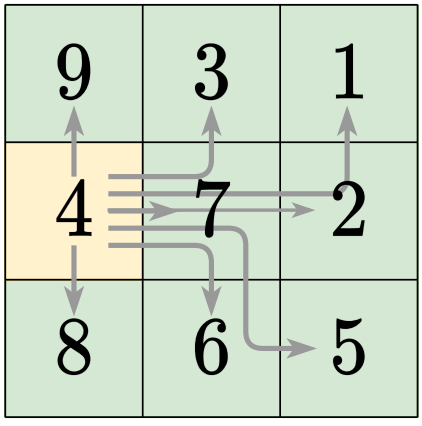


Figure K.1: Visualisation of the third sample case.

Input

The input consists of:

- One line with two integers h and w ($2 \leq h, w \leq 40$), the dimensions of the camping.
- h lines, the ith of which contains w integers $a_{i,1}, \ldots, a_{i,n}$ ($1 \leq a_{i,j} \leq h \cdot w$), the plot numbers. It is guaranteed that all numbers from 1 to $h \cdot w$ occur exactly once.

Output

If there is a position for the reception kiosk such that every other plot can be reached, then output the optimal position for the reception kiosk and the corresponding maximal walking distance. Otherwise, output "`impossible`".

If there are multiple valid solutions, you may output any one of them.

Sample Input 1

```
2 3
1 2 3
6 5 4
```

Sample Output 1

```
2 2
```

Sample Input 2

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

Sample Output 2

```
impossible
```

Sample Input 3

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

Sample Output 3

```
4 3
```

L Lowest Latency

Time limit: 5s

It is the year 2222. The whole universe has been explored, and settlements have been built on every single planet. You live in one of these settlements. While life is comfortable on almost all aspects, there is one dire problem: the latency on the internet connection with other planets is way too high.

Connecting the earth with cables to other planets. Free to use, by PxFuel, modified

Luckily, you have thought of a solution to solve this problem: you just need to put Bonded, Astronomically Paired Cables between all planets, and internet will be super fast! However, as you start developing this idea, you discover that constructing a cable between two planets is more difficult than expected. For this reason, you would like the first prototype of your cable to be between two planets which are as close as possible to each other.

From your astonomy class, you know that the universe is best modelled as a large cube measuring 10^9 lightyears in each dimension. There are exactly 10^5 stationary planets, which are distributed completely randomly through the universe (more precisely: all the coordinates of the planets are independent uniformly random integers ranging from 0 to 10^9).

Given the random positions of the planets in the universe, your goal is to find the minimal Euclidean distance between any two planets.

Input

The input consists of:

- One line with an integer n, the number of planets.
- n lines, each with three integers x, y, and z ($0 \leq x, y, z < 10^9$), the coordinates of one of the planets.

Your submissions will be run on exactly 100 test cases, all of which will have $n = 10^5$. The samples are smaller and for illustration only.

Each of your submissions will be run on new random test cases.

Output

Output the minimal Euclidean distance between any two of the planets.

Your answer should have an absolute or relative error of at most 10^{-6}.

Sample Input 1

```
5
10 5 1
8 2 0
4 7 5
1 0 9
0 10 7
```

Sample Output 1

```
3.7416573867739413
```

Sample Input 2

```
3
790726336 656087587 188785845
976472310 22830435 160538063
211966015 87530388 542618498
```

Sample Output 2

```
660540781.9387681
```