Problem A. Attack on Alpha-Zet

Source file name:	attack.c, attack.cpp, attack.java, attack.py
Input:	`Standard`
Output:	`Standard`

Space pirate Captain Krys has recently acquired a map of the artificial and highly secure planet Alpha-Zet which he has been planning to raid for ages. It turns out the whole planet is built on a 2D plane with modules that serve as one room each. There is exactly one module at every pair of integer coordinates and modules are exactly 1 x 1 units big. Every module is bidirectionally connected to at least one adjacent module. Also, for any two modules there exists exactly one path between them. All in all the modules create a rectangular maze without any loops.

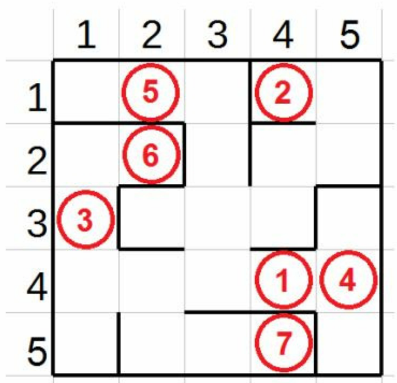


Figure A.1: Illustration of Sample Input 2

On the map Captain Krys has marked several modules he wants to visit in exactly the marked order. What he intends to do there is none of your business, but he promises you a fortune if you determine the number of modules he has to walk through along the route (since there are no loops he will always take the direct route from one marked module to the next). The first marked module indicates where he starts his journey, the last where he wants to finish.

Input

The input consists of:

- one line with two integers h and w ($2 \le h, w \le 1000$) describing the height and the width of the maze.
- $h + 1$ lines follow, describing the maze in ASCII, each line containing $2 \cdot w + 1$ characters. The description always follows these rules:
  - In every row, columns with odd index (starting at index 1) contain either vertical walls or spaces and columns with even index contain either horizontal walls or spaces.
  - The first row describes the northern wall of the maze (which always consists only of horizontal walls). Every subsequent row describes a row of modules.
  - A module is located at every even column index. Its western and eastern walls are located at the directly neighboring odd column indices respectively, its northern wall is located at the same column index but one row above and its southern wall can be found at its own position. If a wall is missing, the corresponding position contains a space instead.
- After the description of the maze, an integer m ($2 \le m \le 10^4$) is given
- Each of the following m lines describes a marked module with two integer coordinates x and y ($1 \le x \le h$; $1 \le y \le w$). The first pair of coordinates is the start point of the journey, the last pair the end point. Modules may appear multiple times but never twice or more in a row. $(1, 1)$ is the top left module and (h, w) is the bottom right module.

It is guaranteed that the maze itself is enclosed. Furthermore it is guaranteed that exactly one path exists between any two modules.

Output

Output one integer, the number of modules Captain Krys has to travel through if he follows the route in the exact order given in the input.

Example

Input	Output
2 6 _ _ _ _ _ _ | _ _ _ _ _| |_ _ _ _ _ _| 5 1 5 1 1 1 6 1 1 1 5	18
5 5 _ _ _ _ _ |_ _ |_ | | _| | _| | |_ _| | | _ _ | |_|_ _ _|_| 7 4 4 1 4 3 1 4 5 1 2 2 2 5 4	43

Problem B. Battle Royale

Source file name:	battle.c, battle.cpp, battle.java, battle.py
Input:	`Standard`
Output:	`Standard`

Battle Royale games are the current trend in video games and Gamers Concealed Punching Circles (GCPC) is the most popular game of them all. The game takes place in an area that, for the sake of simplicity, can be thought of as a two-dimensional plane. Movement and positioning are a substantial part of the gameplay, but getting to a desired location can be dangerous. You are confident in your ability to handle the other players, however, while you are walking to your destination, there are two hazards posed by the game itself:

- The game zone is bounded by a blue circle. Outside of this circle, there is a deadly force field that would instantly take you out of the game.
- Inside the game zone, there is a red circle where you are exposed to artillery strikes. This circle is also too risky to enter

You want to move from one spot on the map to another, but the direct path to your destination is blocked by the red circle, so you need to find a way around it. Can you find the shortest path that avoids all hazards by never leaving the blue or entering the red circle? Touching the boundaries of the circles is fine, as long as you do not cross them.

Input

The input consists of:

- one line with two integers x_c, y_c specifying your current location;
- one line with two integers x_d, y_d specifying your destination;
- one line with three integers x_b, y_b, r_b specifying the center and radius of the blue circle;
- one line with three integers x_r, y_r, r_r specifying the center and radius of the red circle.

All coordinates have an absolute value of at most 1000, and $1 \leq r_b, r_r \leq 1000$. The red circle is strictly inside the blue circle. Your current location and destination are strictly inside the blue circle and strictly outside of the red circle, and the direct path between them is blocked by the red circle.

Output

Output the length of the shortest path that does not leave the blue or enter the red circle. The output must be accurate up to an absolute error of 10^{-7} .

Example

Input	Output

`0 0`
`10 0`
`0 0 1000`
`5 0 2`
	`10.8112187742`

Problem C. Coolest Ski Route

Source file name:	coolest.c, coolest.cpp, coolest.java, coolest.py
Input:	`Standard`
Output:	`Standard`

John loves winter. Every skiing season he goes heli-skiing with his friends. To do so, they rent a helicopter that flies them directly to any mountain in the Alps. From there they follow the picturesque slopes through the untouched snow.

Of course they want to ski on only the best snow, in the best weather they can get. For this they use a combined condition measure and for any given day, they rate all the available slopes.

Can you help them find the most awesome route?

Input

The input consists of:

- one line with two integers n ($2 \leq n \leq 1000$) and m ($1 \leq m \leq 5000$), where n is the number of (1-indexed) connecting points between slopes and m is the number of slopes
- m lines, each with three integers s, t,c ($1 \leq s$, $t \leq n$, $1 \leq c \leq 100$) representing a slope from point s to point t with condition measure c.

Points without incoming slopes are mountain tops with beautiful scenery, points without outgoing slopes are valleys. The helicopter can land on every connecting point, so the friends can start and end their tour at any point they want. All slopes go downhill, so regardless of where they start, they cannot reach the same point again after taking any of the slopes.

Output

Output a single number n that is the maximum sum of condition measures along a path that the friends could take.

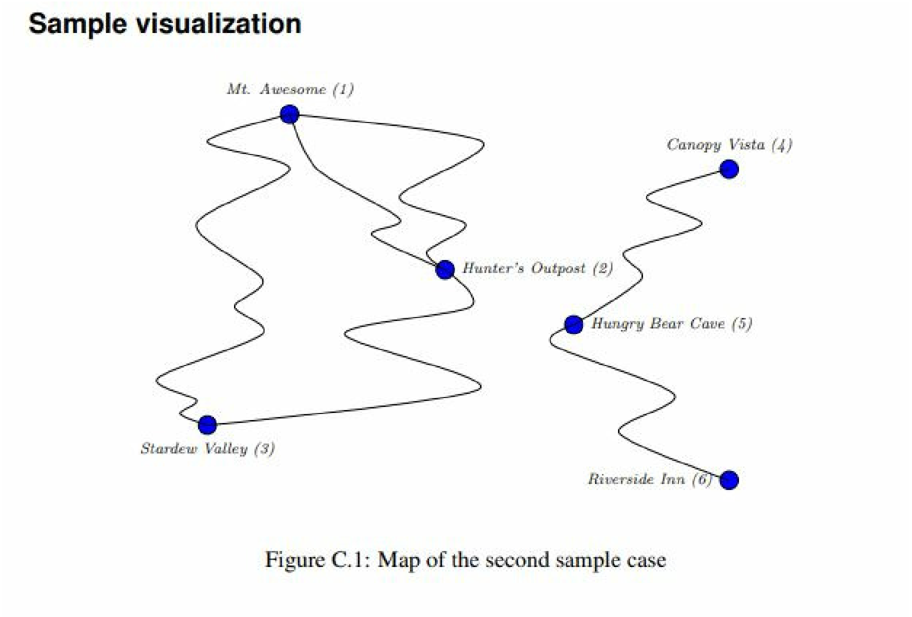


Figure C.1: Map of the second sample case

Example

Input	Output

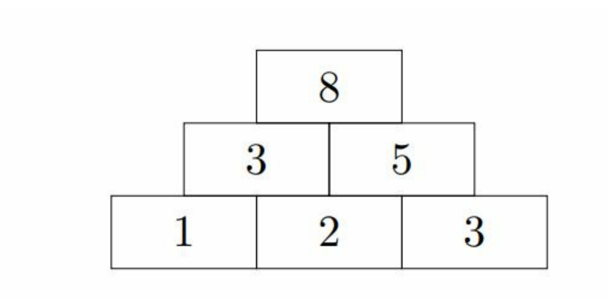
`5 5`
`1 2 15`
`2 3 12`
`1 4 17`
`4 2 11`
`5 4 9`
	`40`

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

Problem D. Down the Pyramid

Source file name:	down.c, down.cpp, down.java, down.py
Input:	`Standard`
Output:	`Standard`

Do you like number pyramids? Given a number sequence that represents the base, you are usually supposed to build the rest of the "pyramid" bottom-up: For each pair of adjacent numbers, you would compute their sum and write it down above them. For example, given the base sequence [1, 2, 3], the sequence directly above it would be [3, 5], and the top of the pyramid would be [8]:



However, I am not interested in completing the pyramid – instead, I would much rather go underground. Thus, for a sequence of n non-negative integers, I will write down a sequence of $n+1$ non-negative integers below it such that each number in the original sequence is the sum of the two numbers I put below it. However, there may be several possible sequences or perhaps even none at all satisfying this condition. So, could you please tell me how many sequences there are for me to choose from?

Input

The input consists of:

- one line with the integer n ($1 \leq n \leq 10^6$), the length of the base sequence.
- one line with n integers $a_1, \ldots, a_n$ ($0 \leq a_i \leq 10^8$ for each i), forming the base sequence.

Output

Output a single integer, the number of non-negative integer sequences that would have the input sequence as the next level in a number pyramid.

Example

Input	Output

`6`
`12 5 7 7 8 4`
	`2`

`3`
`10 1000 100`
	`0`

Problem E. Expired License

Source file name:	expired.c, expired.cpp, expired.java, expired.py
Input:	`Standard`
Output:	`Standard`

Paul is an extremely gifted computer scientist who just completed his master's degree at a prestigious German university. Now he would like to culminate his academic career in a PhD. The problem is that there are so many great universities out there that it is hard for him to pick the best. Because some application deadlines are coming up soon, Paul's only way to procrastinate his decision is by simply applying to all of them.

Most applications require Paul to attach a portrait photo. However, it seems like there does not exist an international standard for the aspect ratio of these kinds of photos. While most European universities ask Paul to send a photograph with aspect ratio 4.5 by 6, some Asian countries discard the applications immediately if the photo does not have an aspect ratio of 7.14 by 11.22, precisely.

As Paul has never been interested in photo editing, he never had a reason to spend a lot of money on proper software. He downloaded a free trial version some months ago, but that version has already expired and now only works with some funny restrictions. The cropping tool, for example, no longer accepts arbitrary numbers for setting the aspect ratio, but only primes. This makes Paul wonder whether the desired aspect ratios can even be properly expressed by two prime numbers. Of course, in case this is possible, he would also like to know the primes he has to enter.

Input

The input consists of:

- one line with an integer n ($1 \leq n \leq 10^5$), the number of applications Paul has to file;
- n lines, each with two real numbers a and b ($0 < a, b < 100$), where $a \times b$ is the desired aspect ratio of one application.

All real numbers are given with at most 5 decimal places after the decimal point.

Output

For each application, if it is possible to represent the desired aspect ratio by two prime numbers p and q, output one line with p and q. Otherwise, output `impossible`. If multiple solutions exist, output the one minimizing $p + q$

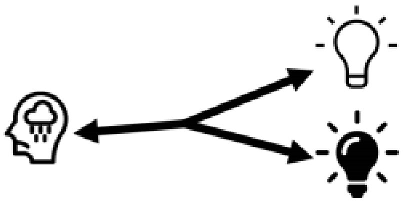
Example

Input	Output

`3`
`4.5 6`
`7.14 11.22`
`0.00002 0.00007`
	`impossible`
	`7 11`
	`2 7`

Problem F. Random Index Vectors

Source file name:	riv.c, riv.cpp, riv.java, riv.py
Input:	`Standard`
Output:	`Standard`



A *Random Index Vector* (RIV) is a data structure that can represent very large arrays where most elements are zero. Here, we consider a version of RIVs which can contain only -1, 0, and $+1$ as elements. There are three basic operations on RIVs:

- Addition of two RIVs a and b; the resulting RIV c is defined by $c_i = a_i + b_i$. The values are clamped at ± 1, *i.e.*, we define $1 + 1 = 1$ and $(-1) + (-1) = -1$.
- Multiplication of two RIVs a and b; the resulting RIV d is defined by $d_i = a_i b_i$.
- Rotation of an RIV a by some integer k; this shifts all of the values in a to the left by k indices. The first k values at the start of a go to the end of the array and become the rightmost values.

An RIV is written in its *condensed form*. This representation is a list that starts with the number of nonzero values, followed by a sorted list of indices (1-indexed) that have nonzero values, where the indices are negated if the values there are -1.

For example, consider an RIV representing an array

$$(1, 0, -1, 0, 0, 0, -1, 0, 0, 1, 0).$$

There are 4 nonzero elements at indices 1, 3, 7 and 10, and the values at 3 and 7 are -1, so the condensed form of this RIV is

$$(4, 1, -3, -7, 10).$$

Given two RIVs in condensed form, add them, multiply them, and rotate them both. Output the results in condensed form.

Input

The first line of input contains two space-separated integers n and k ($1 \le k \le n \le 10^{18}$), where n is the full (uncondensed) length of the RIVs and k is the number of indices to rotate by.

Each of the next two lines contains a condensed form of an RIV, starting with an integer m ($0 \le m \le 1{,}000$), followed by m space-separated indices $i_1, \ldots, i_m$. Each index i_j is a nonzero integer between $-n$ and n (inclusive).

Output

Output four vectors, one per line, in condensed form:

- Sum of the two input RIVs.
- Product of the two input RIVs.

- First RIV rotated by k.
- Second RIV rotated by k.

Example

Input	Output

`30 13`
`6 6 -9 -13 18 22 26`
`8 -1 3 7 11 13 19 20 -27`
	`12 -1 3 6 7 -9 11 18 19 20 22 26 -27`
	`1 -13`
	`6 5 9 13 23 -26 -30`
	`8 6 7 -14 -18 20 24 28 30`

`20 4`
`9 -2 -4 -8 -11 -12 15 18 19 20`
`7 4 5 -10 11 15 18 -20`
	`8 -2 5 -8 -10 -12 15 18 19`
	`5 -4 -11 15 18 -20`
	`9 -4 -7 -8 11 14 15 16 -18 -20`
	`7 1 -6 7 11 14 -16 20`

Problem G. GPS

Source file name:	gps.c, gps.cpp, gps.java, gps.py
Input:	`Standard`
Output:	`Standard`

You, like so many other people, have probably used the Global Positioning Systems (GPS), but have wondered how it actually works. GPS positioning is based on a set of satellites orbiting Earth. All satellites have a synchronized internal clock[1]. They continuously transmit the value of their internal clock to the surface of Earth via radio waves. A GPS receiver simultaneously[2] collects measurements from some satellites which are in sight. The time measurements $t_1, \ldots, t_m$ collected at time t are typically different, as each satellite has a different distance to the receiver. Since we know the exact orbits of all satellites, we know exactly where each satellite was when it sent its signal. Based on these positions and the distance, we can compute our position. To compute a unique position, assuming that we are on the surface of the Earth, we need at least 3 measurements from satellites.

Your task, however, is not to actually compute your own position, but to determine which signals you received from which satellites. To be precise: you are given your geographical coordinates on Earth and a description of all orbits of satellites and should now determine for every satellite

- whether a signal sent from the satellite at a certain time will reach you (which is the case if it does not pass through the Earth).
- if the signal does reach you, the time it took for the signal to travel from the satellite to your position.

Remember that radio waves travel in straight lines and at the speed of light, which we assume is 299 792 458 meters per second. We assume Earth to be a perfect sphere with a radius of 6 371 km, and that all GPS satellites are orbiting the Earth with a fixed speed on perfect circles whose centers are identical to the center of Earth.

Input

The input consists of:

- one line with an integer s $(1 \le s \le 10^4)$, the number of satellites;
- one line with two real numbers l_o, l_a the longitude and latitude of your position;
- s lines, each with four real numbers ϕ, ψ, r, x describing one of the satellites.

For the longitude $l_o \in [-180, 180]$, positive values represent east and negative values west.

For the latitude $l_a \in [-90, 90]$, positive values represent north and negative values south.

- $\phi \in [-180, 180]$ – the reference longitude at which the orbit intersects with the Earth's equator (each orbit intersects the equator (at least) twice). Positive longitude represents east, negative longitude represents west.
- $\psi \in [0, 360]$ - the angle between orbit and the Earth's eastbound equator when passing through the equator at longitude ϕ. Values between 0 and 180 indicate northbound movement while values between 180 and 360 indicate southbound movement. If the orbit is coplanar to the equator, the angle ψ will be 0 or 180.

[1]In practice the clocks are not completely synchronous – because of general relativity. In this problem we assume that these effects do not exist.

[2]Another simplification.

- $r \in [7000, 10^6]$ – the radius of the satellite's orbit in kilometers.
- $x \in [0, 1)$ - the fraction of the orbit the satellite had covered at the time when its signal was sent, starting from its intersection with the equator at ϕ.

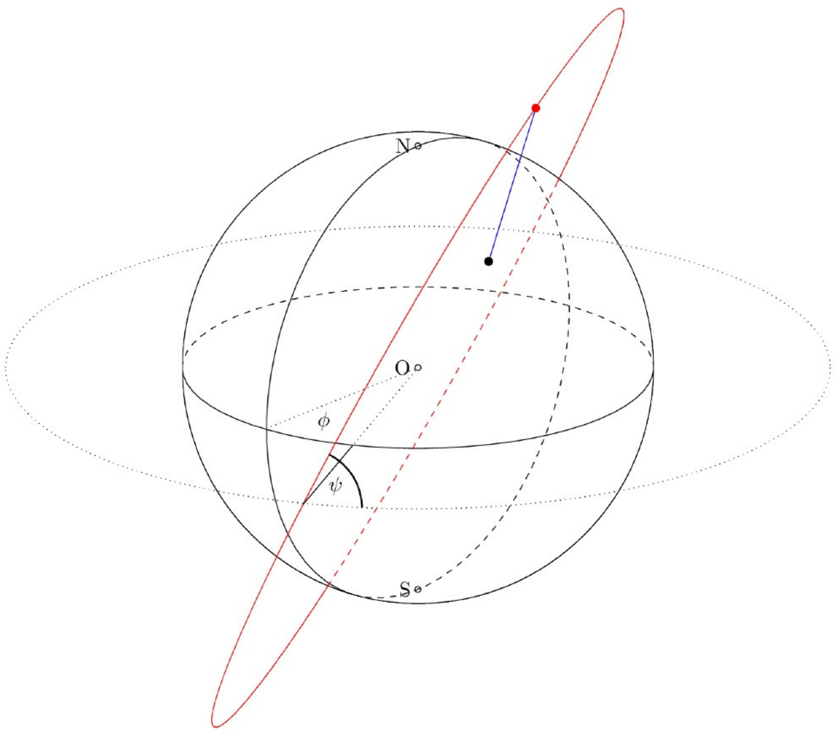


Figure G.1: Schematic display of satellite coordinates. The satellite's orbit is shown in red, its actual position is the red dot. On Earth, the equator and the prime meridian are marked by lines. The dotted line is coplanar to the equator and has the same radius as the satellite's orbit. The angles ϕ and ψ are described in the input section. The black dot is your position on Earth and you should compute the time necessary for the signal to travel the blue line.

Note: All real numbers in the input are given with at most 3 decimal places after the decimal point. It is guaranteed that each satellite has a distance of at least 1 meter from the tangent plane at your position (this means that you can safely use floating point arithmetic in your solution).

Output

For every satellite, output `no signal` if a signal sent from that satellite does not reach you. If the satellite's signal does reach you, output the time that the signal needs to travel from the satellite to your position, in seconds. The time must be accurate up to an absolute error of 10^7.

Example

Input	Output

`3`
`45.0 45.0`
`0.0 45.0 7500.0 0.125`
`0.0 60.0 7000.0 0.3`
`-50.0 5.0 25000.0 0.3`
	`0.00773245`
	`no signal`
	`0.0690288`

Problem H. Pizza Deal

Source file name:	pizza.c, pizza.cpp, pizza.java, pizza.py
Input:	`Standard`
Output:	`Standard`

There's a pizza store which serves pizza in two sizes: either a pizza slice, with area A_1 and price P_1, or a circular pizza, with radius R_1 and price P_2.

You want to maximize the amount of pizza you get per dollar. Should you pick the pizza slice or the whole pizza?

Input

The first line contains two space-separated integers A_1 and P_1.

Similarly, the second line contains two space-separated integers R_1 and P_2.

It is guaranteed that all values are positive integers at most 10^3. We furthermore guarantee that the two will not be worth the same amount of pizza per dollar.

Output

If the better deal is the whole pizza, print `Whole pizza` on a single line.

If it is a slice of pizza, print `Slice of pizza` on a single line.

Example

Input	Output

`8 4`
`7 9`
	`Whole pizza`

`9 2`
`4 7`
	`Whole pizza`

Problem I. It's Time for a Montage

Source file name:	its.c, its.cpp, its.java, its.py
Input:	`Standard`
Output:	`Standard`

The heroes of your favorite action TV show are preparing for the final confrontation with the villains. Fundamentally, there are two rivals who will fight each other: a very important main hero who wants to save the universe and an equally important main villain who wants to destroy it. However, through countless recursive spin-offs, they may have slightly less important sidekicks (a hero and a villain who are rivals themselves), who in turn may also have their own (even less important) sidekicks, and so on. Note that there is an equal number of heroes and villains, and each rival pair has at most one sidekick pair.

Initially, every character will fight their rival, with the winner being determined by who has the higher *Power Level*. If a hero and their corresponding villain have the same Power Level, their battle will be determined by their sidekicks' battle, as the winning sidekick can help as a sort of tiebreaker. (If rivals of equal Power Level do not have sidekicks, the hero character will win with the help of random passersby.) However, whenever a battle is won by either side, there is nothing the sidekicks can do about it – this is because the people behind the show believe some fans might get upset if a character were to get defeated by a bunch of less important characters, so they would lose regardless of the Power Levels.

After the battles between rivals (and possible tiebreakers) are done, the most important character remaining will defeat the rest of the opposing side and determine the fate of the universe. Fortunately, the heroes can ensure victory through hard, rigorous training. For each day they spend training, the Power Level of each hero increases by 1, while the villains' Power Levels remain constant.

But you already knew all this. The question plaguing your mind is how long the training is going to take.

Input

The input consists of:

- one line with an integer n ($1 \leq n \leq 1000$), giving the number of rival pairs.
- one line with n integers $h_1, \ldots, h_n$ ($1 \leq hi \leq 1000$ for each i), the i-th value giving the Power Level of the i-th most important hero.
- one line with n integers $v_1, \ldots, v_n$ ($1 \leq v_i \leq 1000$ for each i), the i-th value giving the Power Level of the i-th most important villain.

Output

Output a single integer, the minimum number of days the heroes need to spend training in order for their side to win.

Example

Input	Output

4
5 3 1 1
8 6 9 1
	4

1 2 1	0

2
4 2
7 5
	3

Problem J. Mountaineers

Source file name:	mountaineers.c, mountaineers.cpp, mountaineers.java, mountaineers.py
Input:	`Standard`
Output:	`Standard`

The Chilean Andes have become increasingly popular as a destination for backpacking and hiking. Many parts of the Andes are quite remote and thus dangerous. Because of this, the Ministry of Tourism wants to help travelers plan their trips. In particular, the travelers need to know how high they will have to climb during their journey, as this information will help them decide which equipment they need to bring. The Ministry has tasked you to provide the aspiring mountaineers with this data

You are given a topographic map of a part of the Andes, represented as a two-dimensional grid of height values, as well as the list of origins and destinations. Mountaineers can move from each grid cell to any of the four adjacent cells. For each mountaineer find the minimal height that they must be able to reach in order to complete their journey

Input

The input consists of:

- one line with three integers m, n and q ($1 \leq m, n \leq 500$, $1 \leq q \leq 10^5$), where m is the number of rows, n is the number of columns, and q is the number of mountaineers;
- m lines, each with n integers $h_1, \ldots, h_n$ ($1 \leq h_i \leq 10^6$), the height values in the map;
- q lines, each with four integers x_1, y_1, x_2, y_2 ($1 \leq x_1, x_2 \leq m$, $1 \leq y_1, y_2 \leq n$), describing a mountaineer who wants to trek from (x_1, y_1) to (x_2, y_2).

The top left cell of the grid has coordinates $(1, 1)$ and the bottom right cell has coordinates (m, n).

Output

Output q integers, the minimal height for each mountaineer, in the same order as in the input.

Example

Input	Output

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

Problem K. Kitchen Cable Chaos

Source file name:	kitchen.c, kitchen.cpp, kitchen.java, kitchen.py
Input:	`Standard`
Output:	`Standard`

You started your new project: installing a home automation system. You already bought all the components and in your local electronic shop you found a promotion set with a bunch of cables of different lengths. Now you want to connect your controller with your smart sandwich maker that is several meters away, but you are lacking a cable that is long enough.

To solve this problem you have to connect some of your cables to form a long one. You measure the lengths of every cable you own. Exactly 5 centimeters of isolation are stripped on both ends of every cable. To connect two cables, you overlap and twist the stripped ends. It is enough for the cables to touch each other with an overlap of 0. You cannot have an overlap of more than 5 centimeters, but the connection quality increases with longer overlaps. On both ends – the controller and the sandwich maker – you also have 5 centimeters of stripped end, to which you have to connect your newly created cable in the same way. The connection quality of your link is determined by the smallest overlap used and your goal is to maximize this value.

The problem would be really easy, but your perfectionist roommate hates unnecessary use of cables. Therefore, the cable has to form a straight line, without any loops or detours. And cutting the cables is no option, obviously.

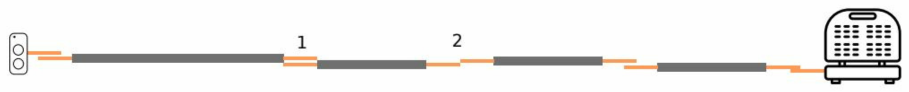


Figure K.1: Four cables of different lengths connect the controller with the sandwich maker, with different overlaps. Connection 1 has the maximal overlap, connection 2 the minimal overlap and all other connections are in between. The quality of this setup is 0.

Considering all possible arrangements of cables, find the one with the best quality

Input

The input consists of:

- one line with two integers n, g ($1 \le n \le 60,\ \ 11 \le g \le 1000$), the number of cables and the distance to be covered in centimeters, measured between the casings of the controller and the sandwich maker;
- n lines, each with an integer d ($11 \le d \le 1000$), giving the lengths of the cables (including the stripped ends).

Each of the n cables can be used at most once

Output

Output one number, the best achievable quality. The quality must be accurate up to an absolute error of 10^{-7}. If no arrangement fits your needs, output `impossible`.

Example

Input	Output
3 70	3.3333333
20	
35	
50	
3 150	impossible
20	
35	
50	