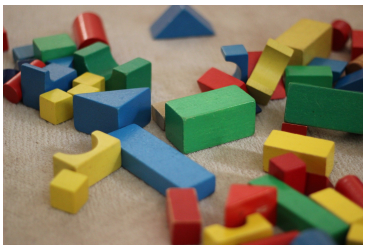
Problem A. Adolescent Architecture

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

Little Peter is building a stack out of his toy blocks. He is using two kinds of blocks – cubes and cylinders – and wants to stack all of them into a single tower, where each block other than the topmost block has a single block standing on it. In order for the tower to be stable, the outline of each block needs to be fully contained within the outline of the block below when looking at the tower from above (the outlines are allowed to touch). Is it possible to construct such a tower, and if so, in which order do the blocks need to be stacked?



Building blocks by Thaliesin on Pixabay

Input

The input consists of:

- One line with an integer n ($1 \leq n \leq 100$), the number of blocks.
- n lines, each with the description of a block. The description consists of a string giving the type of block (`cube` or `cylinder`) and an integer a ($1 \leq a \leq 1\,000$) giving the size of the block – if the block is a cube then a is its side length, and if it is a cylinder then a is the radius of its base (note that the height of the cylinder does not matter).

Output

If there is no way to construct the tower, output `impossible`. Otherwise output n lines, giving the order in which to stack the blocks from top to bottom.

Example

Input	Output
3 cube 7 cube 11 cylinder 5	cube 7 cylinder 5 cube 11
2 cube 5 cylinder 3	impossible
3 cube 4 cylinder 2 cube 4	cylinder 2 cube 4 cube 4

Problem B. Binary Seating

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


By Freepik on www.flaticon.com

By accident, two rooms (room `0` and room `1`) got booked for the theoretical exam of the B++ Applied Programming Course and both were communicated to the students. Now students might go to either of the rooms, and as a student assistant your job is to supervise room `1`. Since you assisted all these students during the course, you know how much time each student will need to finish the exam. Already before the exam you are eager to go home, but you can only leave when all of the students in your examination room have finished. You assume that every student chooses one of the exam rooms with equal probability, independent of the other students. After how much time do you expect to be able to leave?

Input

The input consists of:

- A line with an integer n ($1 \leq n \leq 40$), the number of students.
- A line with n integers $t_1, \ldots, t_n$ ($1 \leq t_i \leq 1000$): t_i is the time it takes for the ith student to finish the exam and leave.

Output

Output the expected time before you can leave. Your answer should have an absolute error of at most 10^{-6}.

Example

Input	Output
2 2 3	2
5 1 4 5 2 3	4.03125
5 2 1 1 1 1	1.46875

Problem C. Cutting Corners

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

CC BY-SA 2.0 by SixRevisions on Flickr

A large coffee spill in the warehouse of the Busy Association of Paper-cutters on Caffeine has stained the corners of all paper in storage. In order to not waste money, it was decided that these dirty corners should be cut off of all pieces of paper.

A few members loudly proclaim that cuts should be made diagonally, while other members say that cutting the corner out as a rectangle is the better option. Both parties claim their method is better.

You decide to end this discussion once and for all, telling the rectangle-cutters that their method is slower. You set out to show them the following: given a piece of paper which has a w by h corner that is stained with coffee that needs to be to cut off, how much more effort is it to cut out the whole rectangle compared to cutting along the diagonal?

Input

The input consists of:

- A line containing two integers w and h ($1 \leq w, h \leq 100$), representing the width and height of the corner respectively.

Output

Output how much longer you have to cut if you cut out a rectangle, compared to cutting along the diagonal. Your answer should have an absolute error of at most 10^{-6}.

Example

Input	Output
3 4	2
12 7	5.107556011
1 1	0.585786438

Problem D. Deck Randomisation

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

CC-BY-SA 4.0 By Alexey Musulev on wikimedia.org

Alice and Bob love playing Don'tminion, which typically involves a lot of shuffling of decks of different sizes. Because they play so often, they are not only very quick at shuffling, but also very consistent. Each time Alice shuffles her deck, her cards get permuted in the same way, just like Bob always permutes his cards the same way when he shuffles them. This isn't good for playing games, but raises an interesting question.

They know that if they take turns shuffling, then at some point the deck will end up ordered in the same way as when they started. Alice shuffles once first, then Bob shuffles once, then Alice shuffles again, et cetera. They start with a sorted deck. What they do not know, however, is how many shuffles it will take before the deck is sorted again.

Can you help them compute how many shuffles it will take? As Alice and Bob can only do 10^{12} shuffles in the limited time they have, any number strictly larger than this should be returned as `huge` instead.

Input

- The first line contains a single integer $1 \leq n \leq 10^5$, the number of cards in the deck.
- The second line contains n distinct integers $1 \leq a_1, a_2, \ldots, a_n \leq n$, where a_i is the new position of the card previously at position i when Alice shuffles the deck.
- The third line contains n distinct integers $1 \leq b_1, b_2, \ldots, b_n \leq n$, where b_i is the new position of the card previously at position i when Bob shuffles the deck.

Output

Output a single positive integer $m > 0$, the minimal number of shuffles required to sort the deck, or `huge` when this number is strictly larger than 10^{12}.

Example

Input	Output
3 2 3 1 3 1 2	2
6 5 1 6 3 2 4 4 6 5 1 3 2	5
8 1 4 2 6 7 8 5 3 3 6 8 4 7 1 5 2	10

Problem E. Exhausting Errands

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

Dolly the delivery drone is out for a busy working day. It has to complete n errands in a street where ℓ houses are lined up in a row, numbered in ascending order as $1, \ldots, \ell$. The distance between adjacent houses is 1. Each errand consists of picking up a package at some house a and delivering it to another house b. Dolly can start with any errand, complete the errands in any order, and is able to carry an unlimited number of packages at the same time. Your job is to find the minimal total distance Dolly has to cover to complete all errands. The delivery route can start and finish at arbitrary locations along the street.

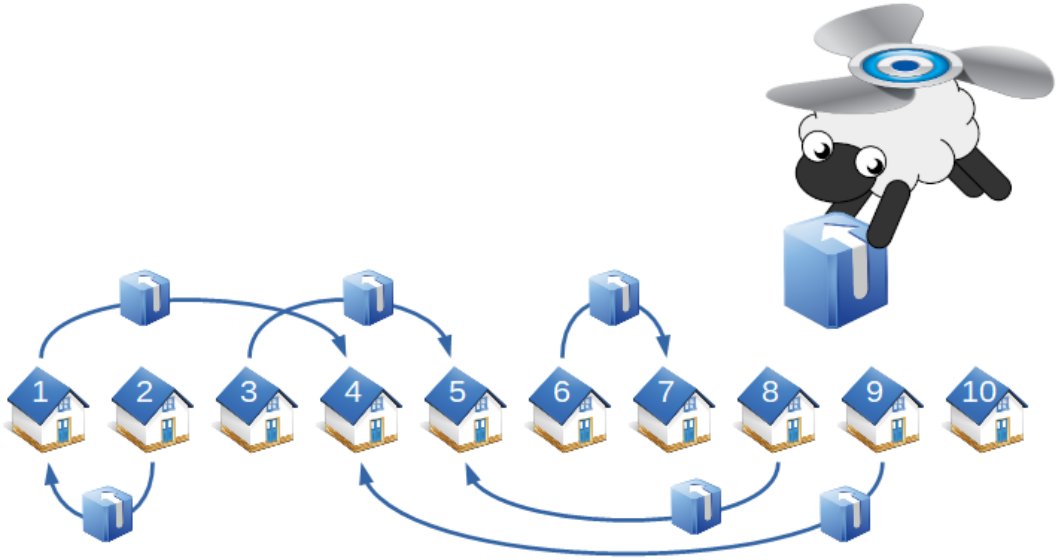


Illustration of Example Input 1. The shortest route is $2 \to 1 \to 9 \to 4$ with length 14.

Input

The input consists of:

- One line with two integers ℓ and n ($1 \leq \ell \leq 10^9$, $1 \leq n \leq 10^5$), where ℓ is the number of houses in the street, and n is the number of errands.
- n lines, each with two integers a and b ($1 \leq a, b \leq \ell$ with $a \neq b$) describing an errand where a package must be picked up at house a and be delivered to house b.

Output

Output a single line with the minimal distance Dolly has to cover from picking up the first package until delivering the last package.

Example

Input	Output
10 6 1 4 3 5 6 7 2 1 9 4 8 5	14
100 3 11 50 50 49 36 35	42

Problem F. Flip Flow

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

Hourglass by nile on Pixabay

For over 600 years, the hourglass has been a well-known timekeeping instrument. An hourglass consists of two glass flasks arranged one above the other which are connected by a narrow channel. Inside the hourglass there is sand which slowly flows from the upper to the lower flask. Hourglasses are typically symmetrical so that the sand always takes the same amount of time to run through, regardless of orientation. For the purposes of this problem, we also assume that the flow rate of the sand is a known constant and does not depend on the amount or distribution of sand in the upper half.

Your friend Helen was bored and has been playing around with her hourglass. At time 0, all the sand was in the lower half. Helen flipped the hourglass over several times and recorded all the moments at which she did so. How many seconds does she need to wait from the current time until all the sand is back in the lower half?

Input

The input consists of:

- One line with three integers t, s and n, where
  - t ($1 \leq t \leq 10^6$) is the current time;
  - s ($1 \leq s \leq 10^6$) is the amount of sand in the hourglass, in grams;
  - n ($1 \leq n \leq 1\,000$) is the number of times the hourglass was flipped.
- One line with n integers $a_1, \ldots, a_n$ ($0 < a_1 < \cdots < a_n < t$), the times at which the hourglass was flipped.

All times are given in seconds. You may assume that the sand flows from the top to the bottom at a constant rate of 1 gram per second.

Output

Output the time in seconds needed for the hourglass to run out starting from time t.

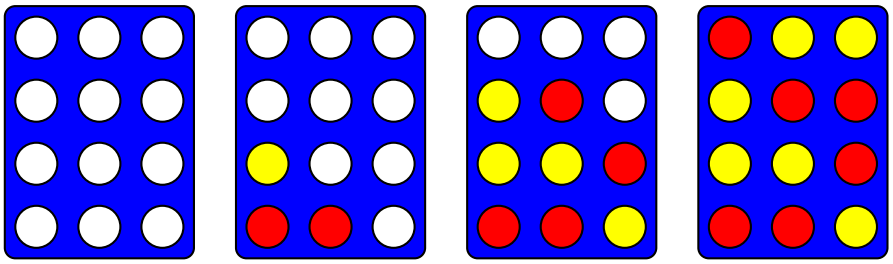
Example

Input	Output
10 7 2 4 9	4
2000 333 3 1000 1250 1500	0
100 10 5 15 20 93 96 97	5

Problem G. Gravity Grid

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

Alice and Bob are playing a generalized version of *Connect Four*. In their game, the board consists of w columns of height h and the goal is to be the first player to complete a row of k tiles of equal colour, either vertically, horizontally or diagonally. The two players alternate dropping their tiles into one of the columns, with Alice using red tiles and going first and Bob using yellow tiles and going second. Once a tile is dropped, it falls down to the bottommost available position, making that position no longer available. Once a column has h tiles in it, it becomes full and the players can no longer drop their tiles there.



Visualisation of the example cases, showing the state of the game after 0, 3, 8 and 12 moves, respectively. Alice's tiles are shown in red, Bob's tiles in yellow.

As Alice and Bob found it quite challenging to keep track of the winning condition, they just kept playing until the board was completely filled with tiles. They recorded a log of the moves they made and asked you to tell them who won the game, and on what turn they did so. If neither player managed to complete a row, the game ends in a draw, so report that instead.

Input

The input consists of:

- One line with three integers h, w and k ($h, w \geq 1, h \cdot w \leq 250\,000, 1 \leq k \leq \max(h, w)$). The columns are numbered from 1 to w.
- One line with $h \cdot w$ integers $a_1, \ldots, a_{h \cdot w}$ ($1 \leq a_i \leq w$ for each i), where a_i is the index of the column that the ith tile was dropped in. Odd indices correspond to Alice's moves and even indices correspond to Bob's moves. Each column appears exactly h times in this list.

Output

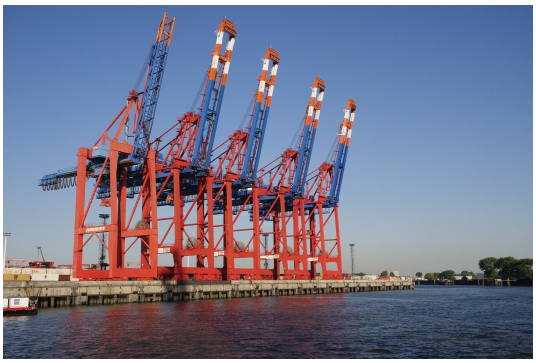
Output the winner of the game (`A` for Alice or `B` for Bob), followed by the number of moves needed to decide the winner. If the game ends in a draw, output `D` instead.

Example

Input	Output
4 3 2 1 1 2 3 3 2 2 1 1 2 3 3	A 3
4 3 3 1 1 2 3 3 2 2 1 1 2 3 3	B 8
4 3 4 1 1 2 3 3 2 2 1 1 2 3 3	D

Problem H. Hectic Harbour

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



Gantry cranes by wasi1370 on Pixabay

There are two gantry cranes operating on the same gantry of length n. The gantry has some fixed integral positions, labelled from 1 to n, at which the cranes must perform loading/unloading operations. In the beginning the first gantry crane is located on the very left of the gantry at position 1, while the second one is located on the very right of the gantry at position n. In each time step a gantry crane can either move to a neighbouring integral position or stay at its current position (and potentially perform a loading/unloading operation). To prevent the gantry cranes from hitting each other, the first crane needs to stay strictly to the left of the second crane at all times. For both cranes you are given a task list consisting of gantry positions at which the cranes must perform loading/unloading operations. Both cranes must perform their assigned operations in the given order. What is the minimal amount of time necessary for both gantry cranes to finish their tasks? It is guaranteed that the first gantry crane never has to operate at position n of the gantry while the second gantry crane never has to operate at position 1. For both gantry cranes the first and last loading/unloading operation in the task list is their initial position.

Input

The input consists of:

- One line with integers n, a and b where
  - n ($2 \leq n \leq 2\,000$) is the length of the gantry;
  - a ($2 \leq a \leq 50$) is the number of operations in the task list of the first gantry crane;
  - b ($2 \leq b \leq 50$) is the number of operations in the task list of the second gantry crane.
- One line with a integers $k_1, \ldots, k_a$ ($1 \leq k_i \leq n-1$ for all i), the tasks of the first gantry crane.
- One line with b integers $\ell_1, \ldots, \ell_b$ ($2 \leq \ell_i \leq n$ for all i), the tasks of the second gantry crane.

The first and last task of both gantry cranes are at their initial position, i.e., $k_1 = k_a = 1$ and $\ell_1 = \ell_b = n$.

Output

Output the minimum number of time steps necessary for both gantry cranes to finish their assigned tasks.

Example

Input	Output
3 2 4 1 1 3 3 2 3	6
4 4 4 1 2 3 1 4 3 3 4	9

Explanation

In the first example test case the gantry is of length 3, the first gantry crane has 2 operations in its task list while the second gantry crane has 4 operations in its task list. At least 6 time steps are necessary for both gantry cranes to finish their assigned tasks.

Time	Gantry Crane 1	Gantry Crane 2
1	Operate at 1	Operate at 3
2	Operate at 1	Operate at 3
3	Idle at 1	Move from 3 to 2
4	Idle at 1	Operate at 2
5	Idle at 1	Move from 2 to 3
6	Idle at 1	Operate at 3

In the second example test case the gantry is of length 4 and both gantry cranes have to perform 4 operations. At least 9 time steps are necessary for both gantry cranes to finish their assigned tasks.

Time	Gantry Crane 1	Gantry Crane 2
1	Operate at 1	Operate at 4
2	Move from 1 to 2	Move from 4 to 3
3	Operate at 2	Operate at 3
4	Move from 2 to 3	Move from 3 to 4
5	Operate at 3	Idle at 4
6	Move from 3 to 2	Move from 4 to 3
7	Move from 2 to 1	Operate at 3
8	Operate at 1	Move from 3 to 4
9	Idle at 1	Operate at 4

Problem I. Efficient Exchange

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

You have recently acquired a new job at the Bank for Acquiring Peculiar Currencies. Here people can make payments, and deposit or withdraw money in all kinds of strange currencies. At your first day on the job you help a customer from Nijmegia, a small insignificant country famous for its enormous coins with values equal to powers of 10, that is, $1, 10, 100, 1000$, etc. This customer wants to make a rather large payment, and you are not looking forward to the prospect of carrying all those coins to and from the vault.

You therefore decide to think things over first. You have an enormous supply of Nijmegian coins in reserve, as does the customer (most citizens from Nijmegia are extremely strong). You now want to minimize the total number of coins that are exchanged, in either direction, to make the exact payment the customer has to make.

For example, if the customer wants to pay 83 coins there are many ways to make the exchange. Here are three possibilities:

Option 1. The customer pays 8 coins of value 10, and 3 coins of value 1. This requires exchanging $8 + 3 = 11$ coins.

Option 2. The customer pays a coin of value 100, and you return a coin of value 10, and 7 coins of value 1. This requires exchanging $1 + 1 + 7 = 9$ coins.

Option 3. The customer pays a coin of value 100, and 3 coins of value 1. You return 2 coins of value 10. This requires exchanging $1 + 3 + 2 = 6$ coins.

It turns out the last way of doing it requires the least coins possible.

Input

A single integer $0 \leq n < 10^{1000}$, the amount the customer from Nijmegia has to pay.

Output

Output the minimum number of coins that have to be exchanged to make the required payment.

Example

Input	Output
83	6
13	4
0	0
12345678987654321	42

Problem J. Jeopardised Journey

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

Little Red Riding Hood's grandmother's hut is already quite old. It needs to be torn down and will be replaced with a new hut. Naturally, she wants the hut to be as safe as possible – nothing could be worse than Little Red Riding Hood being eaten by the big bad wolf.

Grandmother wants to build her hut in one of the forest's glades. Little Red Riding Hood's house – well, that of her parents – is also in a glade. Furthermore, the wolf is also always lurking in one of the glades. If Little Red Riding Hood walks from her home to Grandmother's hut and passes by the wolf, she will surely be eaten. The wolf only changes its glade during the night, so whenever Little Red Riding Hood is walking through the forest, he will be located in an unknown but fixed glade. He will never be in the glade of Grandmother's hut nor in the glade of Little Red Riding Hood's house.

Little Red Riding Hood always walks from one glade to another. In order to avoid getting eaten by the wolf, she has to check whether the wolf is in the glade she currently wants to walk to. Luckily, she has a pair of binoculars. So whenever she is in one of the glades and wants to go to some other glade, she first uses her binoculars to check whether the wolf is in that other glade. If he is, she won't go to that glade. The problem, however, is the following: The forest is not flat. Throughout the forest there are hills, and if a hill is in-between Little Red Riding Hood's current glade and the next glade she wants to go to, she cannot use her binoculars to check whether the wolf is in that next glade. She will never go from her current glade to another glade if she cannot check that other glade with her binoculars. This includes both the glade of her own home and that of Grandmother's hut (the wolf will never be there, but it is still important for her to check in order to feel comfortable). Each of the hills is a perfect circle and we assume that the glades are sufficiently small compared to the hills so that we consider them to be points. Even if the line-of-sight between two glades is only tangent to a hill, Little Red Riding Hood's sight is blocked. No two hills intersect and no glade is within or on the boundary of a hill.

Grandmother considers a glade safe for her to build her new hut there if no matter in which glade the wolf sits, Little Red Riding Hood can always find a way from her home to Grandmother and back. Grandmother has asked you to determine which of the glades are safe for her to build her house in.

Input

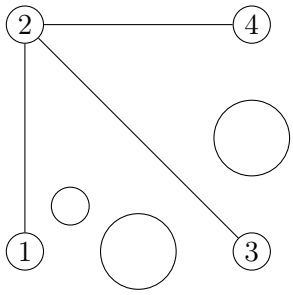
The input consists of:

- One line with two integers g and h ($2 \leq g \leq 2\,000$, $0 \leq h \leq 2\,000$), the number of glades and the number of hills. The glades are numbered from 1 to g. Little Red Riding Hood's house is always located in the gth glade.
- g lines, where the ith line contains the two integers x_i and y_i ($-10^7 \leq x_i, y_i \leq 10^7$) giving the ith glade's coordinates.
- h lines, each with three integers x, y, and r ($-10^7 \leq x, y \leq 10^7$, $0 < r \leq 10^7$) giving the centre and radius of one of the hills.

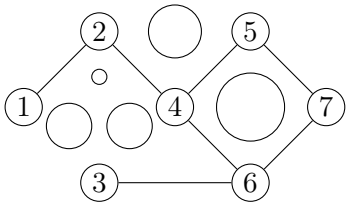
No two glades have the same coordinates, no two hills share a common point, and no glade is within or on the boundary of a hill.

Output

Output the indices of all glades where it is safe for Grandmother to build her hut. These indices must be output in ascending order.



The first example input.



The second example input.

Example

Input	Output
4 3 0 0 0 10 10 0 10 10 5 0 2 10 5 2 2 2 1	2
7 5 0 0 10 10 10 -10 20 0 30 10 30 -10 40 0 10 4 1 6 -3 3 14 -3 3 20 10 4 30 0 4	4 5 6

Problem K. Kangaroo Commotion

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

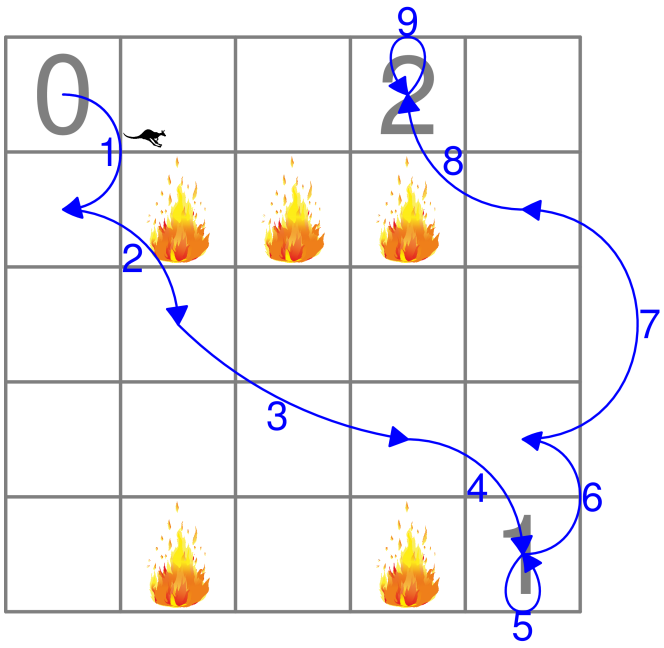
Bushfires are threatening your habitat! Being a kangaroo, you must inform the other kangaroos in your troop as fast as possible and flee to a safe area.

You are currently standing still, and you will jump to the other kangaroos' locations. You will visit the other kangaroos in a specific order so that all of them have sufficient time to escape. After visiting all other kangaroos you must continue jumping to the safe area, where you should come to a stop.

With each jump you move a (possibly negative) integer distance north and/or east. Because of your limited muscle power, you are only able to accelerate or decelerate at most 1 in each direction each jump. Formally, if jump i moves you $v_{x,i}$ to the north and $v_{y,i}$ to the east, the next jump $i+1$ must satisfy $|v_{x,i+1} - v_{x,i}| \leq 1$ and $|v_{y,i+1} - v_{y,i}| \leq 1$.

Find the minimal number of jumps needed to go from your current position to the safe area via all other kangaroos, without leaving the grid. It is very important to come to a full stop at the end, so the last jump must both start and end at the safe area.

The first example is shown in the next figure.



Visualisation of example 1 showing one possible way to get to the safe area using 9 jumps.

Input

The input consists of:

- A line containing three integers r, c ($1 \leq r, c \leq 50$), the number of rows and columns of the grid, and k ($1 \leq k \leq 5$), the number of other kangaroos you need to warn.
- r lines each consisting of c characters. A "." indicates an open space and a "#" indicates a bush where you can't jump.

Your start position is indicated by a "0" and the characters "1" to k indicate the positions of the other kangaroos you need to warn *in this order*.

The position indicated by $k+1$ indicates the safe area where you should come to a stop.

Output

If it is possible to reach the safe area, then output the minimal number of steps needed to reach the safe area. Otherwise, output "impossible".

Example

Input	Output
5 5 1 0..2. .###. ....#.#1	9
2 2 2 03 12	4
1 5 1 .0#21	8
3 4 1 #0## #.#2 1###	impossible

Problem L. Find my Family

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

CC-BY 2.0 By Ivan on Flickr

You are looking for a particular family photo with you and your favorite relatives Alice and Bob. Each family photo contains a line-up of n people. On the photo you're looking for, you remember that Alice, who is taller than you, was somewhere on your left from the perspective of the photographer. Also, Bob who is taller than both you and Alice, was standing somewhere on your right.

Since you have a large number of family photos, you want to use your computer to assist in finding the photo. Many of the photos are quite blurry, so facial recognition has proven ineffective. Luckily, the Batch Apex Photo Classifier, which detects each person in a photo and outputs the sequence of their (distinct) heights in pixels, has produced excellent results. Given this sequence of heights for k photos, determine which of these photos could potentially be the photo you're looking for.

Input

- The first line contains $1 \leq k \leq 1000$, the number of photos you have to process.
- Then follow two lines for each photo.
  - The first line contains a single integer $3 \leq n \leq 3 \cdot 10^5$, the number of people on this photo.
  - The second line contains n distinct integers $1 \leq h_1, \ldots, h_n \leq 10^9$, the heights of the people in the photo, from left to right.

It is guaranteed that the total number of people in all photos is at most $3 \cdot 10^5$.

Output

- On the first line, output the number of photos k that need further investigation.
- Then print k lines each containing a single integer $1 \leq a_i \leq n$, the sorted indices of the photos you need to look at.

Example

Input	Output
1 3 2 1 3	1 1
4 4 140 157 160 193 5 15 24 38 9 30 6 36 12 24 29 23 15 6 170 230 320 180 250 210	2 2 4

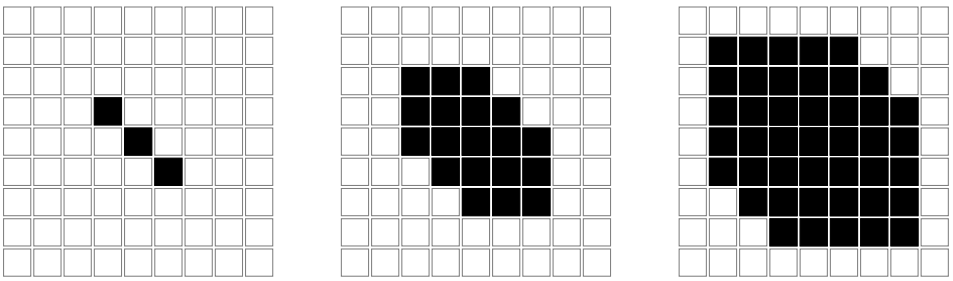
Problem M. Gluttonous Goop

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

As a prominent researcher in the laboratorium for Breathtaking Agriculture through Petri dish Cultivation, you keep on looking for new organisms that might be the food of the future. Recently you have discovered a new fungus-type organism that seems to be nutritious and very efficient in converting energy from food to body mass. You have placed a small batch surrounded by food in a petri dish and watched it grow for a bit.

However, now that the weekend has arrived, you would rather spend some time with your loved ones than stare at the contents of this petri dish all the time (even though it is a *fun guy*). You cannot leave without taking the necessary precautions. What if the fungus grows too large and starts eating away at the rest of the laboratory?!

You model the situation as follows: you divide the plane into 1×1-squares, and draw where the fungus currently is. You know that every time step, if the fungus occupies a square, it will expand to all eight neighbouring squares (and still occupy the initial square). You know how many time steps you will be gone for over the weekend, and now you want to know how many squares the fungus will occupy when you get back.



Example of fungus growth: the fungus from sample 2 after 0, 1, 2 time steps. The middle image corresponds to the correct output for example 2.

N.B.: In the input, the fungus will be given on a finite grid, but it can (and will!) grow beyond these boundaries. The fungus is not so easily contained.

Input

- First a line containing integers $1 \leq r, c \leq 20$, and $0 \leq k \leq 10^6$, denoting the number of rows and columns of the initial grid and the number of time steps.
- Then follow r lines of c characters, each character being either '`.`' or '`#`'. A '`#`' denotes that the fungus is occupying this square. The fungus need not be connected.

Output

Output the number of squares the fungus occupies after k time steps have passed.

Example

Input	Output
5 5 3 ..... .###. .#.#. .###.	81
3 3 1 #.. .#. ..#	19
4 6 3 ..##.. .#..#. .#..#. ..##..	96
1 1 1000000 #	4000004000001

Problem N. Gaggle

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

At the new start-up company Gaggle, we have rejected the oppressive corporate structures of old, with all of their managers and subordinates and hierarchies and so on. Instead we have embraced a free and open corporate culture in which all employees (called Gagglers) are in charge of themselves and allowed to roam free.

Rather than having managers overseeing the work, the main method used to coordinate work at Gaggle is a mentor system: each Gaggler designates some other Gaggler as their mentor, with whom they discuss their ongoing projects. This mentor relation may or may not be symmetric (in other words you may or may not be the mentor of your mentor) but you can never be the mentor of yourself.

Initially, all Gagglers were able to pick anyone they liked as their mentor, but after a while it was discovered that this lead to two problems:

1. Some people were more popular than others and had too many choosing them as their mentor, causing them not to have time to do their actual work.
2. Some flocks of Gagglers ended up isolated from the rest of the company (e.g., if Gagglers A and B are each other's mentors and they are not the mentor of anyone else), causing failure of these flocks to coordinate with the rest of the company.

In order to remedy these two flaws, it was (collectively) decided that:

1. Every Gaggler must be the mentor of exactly one other Gaggler, and
2. Assuming every Gaggler only communicates with their mentor and their mentee, it must still be possible for any information that any Gaggler has to reach any other Gaggler.

In order to reward lower-numbered (more senior) Gagglers while introducing this new policy, it was decided that lower-numbered Gagglers should get to keep their current mentor if possible, and if they have to change, their new mentor should be as low-numbered (more senior, and therefore more experienced) as possible.

Concretely, consider two possible new assignments of mentors, and suppose the lowest-numbered Gaggler where these assignments differ is Gaggler number i. Then if one of the two assignments assigns Gaggler i the same mentor as they originally had, we prefer that assignment. Otherwise, if Gaggler i gets a new mentor in both of the two assignments, then we prefer the assignment where the number of the new mentor of Gaggler i is smaller.

For example, consider Sample Input 2 below. One possible new assignment of mentors would be to simply change so that Gaggler 1 becomes mentored by Gaggler 2. However, in the best assignment, shown in Sample Output 2, we let Gaggler 1 keep their current mentor and instead change the mentors of both Gagglers 2 and 3.

Input

The first line of input contains a single integer n ($2 \leq n \leq 500\,000$), the number of Gagglers. Then follows a line containing n integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq n$ and $a_i \neq i$ for each i) where a_i is the current mentor of Gaggler i (the Gagglers are numbered from 1 to n).

Output

Then output a line with the new assignment $b_1, \ldots, b_n$ of mentors, in the same format as in the input. The new list should be a valid assignment according to the new requirements, and be the best according to the tie-breaking rule described above.

Example

Input	Output
4 2 1 4 3	2 3 4 1
3 3 3 1	3 1 2

Problem O. Pitch Performance

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

After a recent disaster at the Easter party karaoke, you are working on improving your singing. To gauge how well you are doing, you would like to measure how much the pitch and timing of your singing differs from the target melody you were trying to perform.

We model the melody in a simplified manner as a piecewise-constant function f, where at time x the melody has pitch $f(x)$. In other words from time 0 up to some time x_1, $f(x)$ is some constant value y_1, and then at time x_1 it changes to some other value y_2 and remains at that value until some time $x_2 > x_1$, and so on.

Your voice, on the other hand, is of a more wavering nature, and you may generally not be able to hold an exact constant pitch for any period of time, sometimes breaking off into an unwelcome falsetto and sometimes croaking on those low tones. The pitch of your voice can be modeled in a highly simplistic way as a piecewise-quadratic function g. In other words from time 0 up to x_1 (not necessarily the same x_1 as for the function f), your pitch $g(x)$ agrees with some quadratic polynomial, and then from time x_1 to x_2 with some other quadratic polynomial, and so on.

The difference between your performance g and the target melody f is the area between these two functions. See Figure for an example. Given the melody f and your performance g, compute their difference.

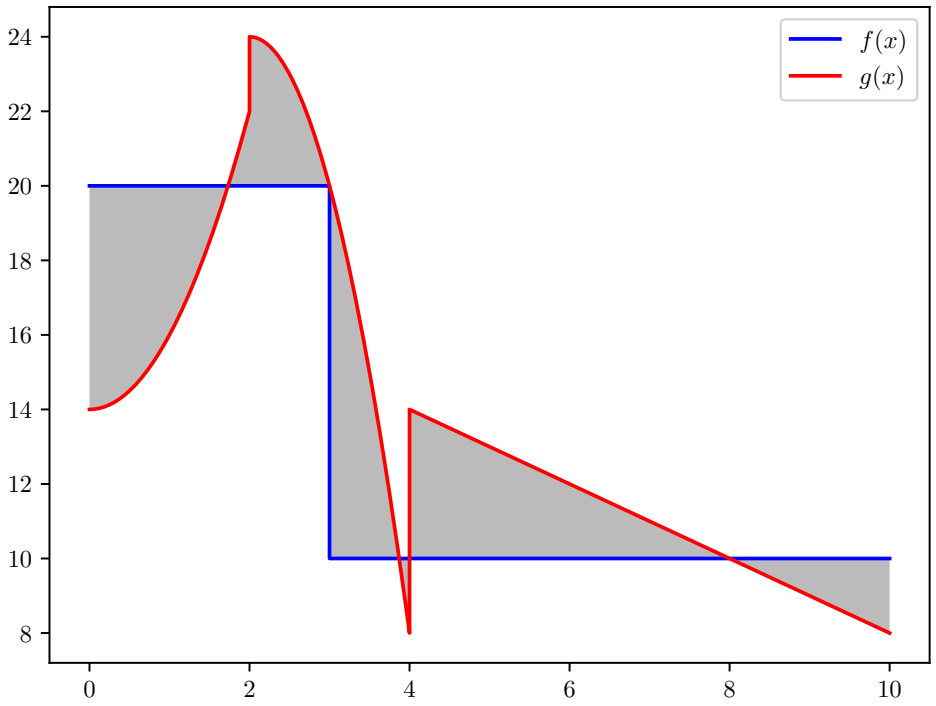


Illustration of Example Input 1. The difference between f and g is the area of the shaded region in the figure.

Input

The first line of input contains an integer n ($1 \leq n \leq 500$), the number of pieces in the target melody function f. Then follow n lines describing f. The i'th such line contains two integers x_i and y_i ($x_{i-1} < x_i \leq 10^4$

and $0 \leq y_i \leq 10^4$). For all x in the half-open interval $[x_{i-1}, x_i)$, the value of $f(x)$ equals y_i. For the first interval we define $x_0 = 0$.

Then follows a line containing an integer m ($1 \leq m \leq 500$), the number of pieces in the function g describing your performance. The next m lines contain the description of g. The i'th such line contains four integers x'_i, a_i, b_i and c_i ($x'_{i-1} < x'_i \leq 10^4$ and $-10^7 \leq a_i, b_i, c_i \leq 10^7$). For all x in the half-open interval $[x'_{i-1}, x'_i)$, the value of $g(x)$ equals $a_i x^2 + b_i x + c_i$. For the first interval we define $x'_0 = 0$.

You may assume that $0 \leq g(x) \leq 10^4$ for all $x'_0 \leq x \leq x'_m$ and that the two functions end at the same time (i.e., $x_n = x'_m$).

Output

Output the difference between f and g. Your output should be correct to within an absolute error of at most 10^{-6}.

Example

Input	Output
2 3 20 10 10 3 2 2 0 14 4 -4 16 8 10 0 -1 18	24.7785420704411
1 20 50 1 20 1 -20 100	609.47570824873

Problem P. Proofs

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

You are teaching discrete math. You have done your best to teach your students about axioms and inference rules, proofs and theorems. Sometimes the students write beautiful proofs that Fermat would be proud of but sometimes, also like Fermat, their proofs are not quite right. You are getting a little tired of hunting through some of these so-called "proofs" for the magic tricks that let them prove $1 = 2$ and had the great idea to write a computer program to speed things up!

Because this is the first class in proof-based mathematics, you have started your students off with a simple proof system. All proof lines consist of a list of assumptions, an arrow, and a conclusion. If there are no assumptions, the conclusion is an axiom. A line of the proof is valid if and only if all assumptions were conclusions of previous lines. Sometimes the students derive a conclusion more than once just to be extra sure it is true, and that is perfectly all right!

Input

The first line of input consists of an integer $1 \leq n \leq 400\,000$, the number of lines in the "proof". Then follow the n lines of the "proof". Each line has $0 \leq a \leq 5$ assumptions, followed by an arrow (the string "`->`"), followed by one conclusion. All assumptions and conclusions consist of $1 \leq c \leq 5$ uppercase alphabetic characters. The assumptions, arrow, and conclusion are all separated by single spaces.

Output

If every line is correct output "`correct`". Otherwise, output the number of the first line with an error (line numbers start at 1).

Example

Input	Output
3 -> ALICE -> BOB ALICE BOB -> CARL	correct
1 A -> B	1

Problem Q. Triple Texting

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

Julia enjoys talking to her grandma, playing with legos, and inventing two-player card games where she has a winning strategy. Recently however, she has not been able to talk to her grandma in person because of some kind of "pandemonium". Instead, they have resorted to texting, which is a very slow process since grandma types very slowly and often mistypes letters. To make matters worse, grandma has started to write every word three times so that Julia can correct her mistypes. For example, if grandma wants to write the word "hello", she will instead write "hellohellohello". If she mistypes one of those letters, it might instead be sent as "hellohrllohello'.'

Your task is to write a program that given a message sent by grandma, where possibly one letter has been changed to some other letter, finds the original word.

Input

The input consists of one string s containing lower case English letters ($3 \leq |s| \leq 99$). This is the message sent by grandma. It is guaranteed that this string is the result of a word being written three times, where possibly one letter was changed to some other letter.

Output

Output one string t, the original word.

Example

Input	Output
`hellohrllohello`	`hello`
`hejhejhej`	`hej`

Problem R. Winning the Vote

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

In the country of Elecuador, a very strange voting system is used. When it is time for the election, each one of the n citizens will arrive in some order to the voting station. There are only two parties to vote for, conveniently named 1 and 2. When arriving to the voting station, a person will vote for one of the parties, unless they are a *teller*. The tellers do not vote, instead they count how many votes each of the two parties has at the time the teller arrives, and if one of the parties has more votes than the other then that party receives one point (if the two parties have the same number of votes, neither of them receives a point). The party with the most points at the end wins. If both parties end up with the same number of points, chaos ensues.

As the president of Elecuador representing party 1, you are worried that the coming election will be the end of your reign. Fortunately, you have a plan to stop this from happening. Being the president, you know who everyone in the country will vote for, who the tellers are, and in what order everyone will arrive to the voting station. By making the right phone calls, you can also affect when the tellers arrive. In one move, it is possible to swap a teller with an adjacent person in the list of arrivals to the voting station. Note that it is not possible to swap two adjacent non-tellers. What is the minimum number of swaps necessary to ensure that party 1 wins?

Input

The input starts with a line containing an integer n ($1 \leq n \leq 5\,000$), the number of citizens in Elecuador. Then follows a line containing a string s of length n, consisting of the characters 0, 1, and 2. This string represents the citizens in the order they arrive to the voting station. If the i'th character s_i is 1 or 2, it means that the i'th citizen will vote for party 1 or 2, respectively. If s_i is 0, it means that the i'th citizen is a teller.

Output

If it is possible to ensure victory, output one integer, the minimum number of swaps necessary. Otherwise, output "`impossible`".

Example

Input	Output
8 12210020	4
4 1111	impossible
11 00211222220	5