Problem A. Artwork

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

A template for an artwork is a white grid of $n \times m$ squares. The artwork will be created by painting q horizontal and vertical black strokes. A stroke starts from square (x_1, y_1), ends at square (x_2, y_2) ($x_1 = x_2$ or $y_1 = y_2$) and changes the color of all squares (x, y) to black where $x_1 \leq x \leq x_2$ and $y_1 \leq y \leq y_2$.

The beauty of an artwork is the number of regions in the grid. Each region consists of one or more white squares that are connected to each other using a path of white squares in the grid, walking horizontally or vertically but not diagonally. The initial beauty of the artwork is 1. Your task is to calculate the beauty after each new stroke. Figure 1 illustrates how the beauty of the artwork varies in Sample Input 1.

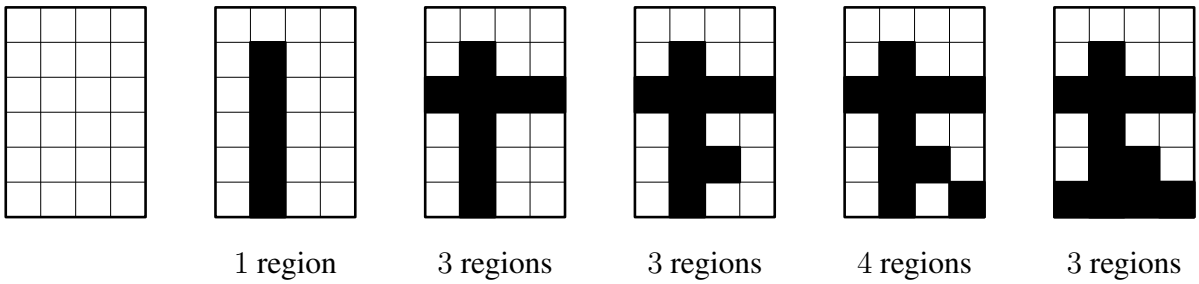


Illustration of Sample Input 1.

Input

The first line of input contains three integers n, m and q ($1 \leq n, m \leq 1000$, $1 \leq q \leq 10^4$).

Then follow q lines that describe the strokes. Each line consists of four integers x_1, y_1, x_2 and y_2 ($1 \leq x_1 \leq x_2 \leq n$, $1 \leq y_1 \leq y_2 \leq m$). Either $x_1 = x_2$ or $y_1 = y_2$ (or both).

Output

For each of the q strokes, output a line containing the beauty of the artwork after the stroke.

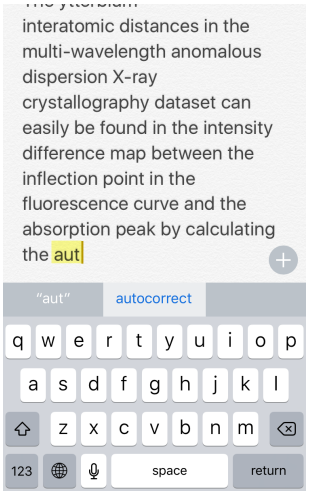
Example

Input	Output
4 6 5 2 2 2 6 1 3 4 3 2 5 3 5 4 6 4 6 1 6 4 6	1 3 3 4 3

Problem B. Bless You Autocorrect!

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

Typing on phones can be tedious. It is easy to make typing mistakes, which is why most phones come with an autocorrect feature. Autocorrect not only fixes common typos, but also suggests how to finish the word while you type it. Jenny has recently been pondering how she can use this feature to her advantage, so that she can send a particular message with the minimum amount of typing.



The autocorrect feature on Jenny's phone works like this: the phone has an internal dictionary of words sorted by their frequency in the English language. Whenever a word is being typed, autocorrect suggests the most common word (if any) starting with all the letters typed so far. By pressing tab, the word being typed is completed with the autocorrect suggestion. Autocorrect can only be used after the first character of a word has been typed – it is not possible to press tab before having typed anything. If no dictionary word starts with the letters typed so far, pressing tab has no effect.

Jenny has recently noticed that it is sometimes possible to use autocorrect to her advantage even when it is not suggesting the correct word, by deleting the end of the autocorrected word. For instance, to type the word "autocorrelation", Jenny starts typing "aut", which then autocorrects to "autocorrect" (because it is such a common word these days!) when pressing tab. By deleting the last two characters ("ct") and then typing the six letters "lation", the whole word can be typed using only 3 ("aut") +1 (tab) +2 (backspace twice) +6 ("lation") = 12 keystrokes, 3 fewer than typing "autocorrelation" without using autocorrect.

Given the dictionary on the phone and the words Jenny wants to type, output the minimum number of keystrokes required to type each word. The only keys Jenny can use are the letter keys, tab and backspace.

Input

The first line of input contains two positive integers n ($1 \leq n \leq 10^5$), the number of words in the dictionary, and m ($1 \leq m \leq 10^5$), the number of words to type. Then follow n lines with one word per line, sorted in decreasing order of how common the word is (the first word is the most common). No word appears twice in the dictionary. Then follow m lines, containing the words to type.

The dictionary and the words to type only use lower case letters '`a`'-'`z`'. The total size of the input file is at most 1 MB.

Output

For each word to type, output a line containing the minimum number of keystrokes required to type the corresponding word.

Example

Input	Output
5 5 austria autocorrect program programming computer autocorrelation programming competition zyx austria	12 4 11 3 2
5 3 yogurt you blessing auto correct bless you autocorrect	5 3 9

Problem C. Card Hand Sorting

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

When dealt cards in the card game Plump it is a good idea to start by sorting the cards in hand by suit and rank. The different suits should be grouped and the ranks should be sorted within each suit. But the order of the suits does not matter and within each suit, the cards may be sorted in either ascending or descending order on rank. It is allowed for some suits to be sorted in ascending order and others in descending order.

Sample Input 2 (cc by-sa NCPC 2016)

Sorting is done by moving one card at a time from its current position to a new position in the hand, at the start, end, or in between two adjacent cards. What is the smallest number of moves required to sort a given hand of cards?

Input

The first line of input contains an integer n ($1 \leq n \leq 52$), the number of cards in the hand. The second line contains n pairwise distinct space-separated cards, each represented by two characters. The first character of a card represents the rank and is either a digit from 2 to 9 or one of the letters `T`, `J`, `Q`, `K`, and `A` representing Ten, Jack, Queen, King and Ace, respectively, given here in increasing order. The second character of a card is from the set {`s`, `h`, `d`, `c`} representing the suits spades ♠ , hearts ♡, diamonds ♢, and clubs ♣.

Output

Output the minimum number of card moves required to sort the hand as described above.

Example

Input	Output
4 2h Th 8c Qh	1
7 9d As 2s Qd 2c Jd 8h	2
4 2h 3h 9c 8c	0

Problem D. Daydreaming Stockbroker

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

Gina Reed, the famous stockbroker, is having a slow day at work, and between rounds of solitaire she is daydreaming. Foretelling the future is hard, but imagine if you could just go back in time and use your knowledge of stock price history in order to maximize your profits!

Photo by liz west on flickr, cc by

Now Gina starts to wonder: if she were to go back in time a few days and bring a measly $100 with her, how much money could she make by just buying and selling stock in Rollercoaster Inc. (the most volatile stock in existence) at the right times? Would she earn enough to retire comfortably in a mansion on Tenerife?

Note that Gina can not buy fractional shares, she must buy whole shares in Rollercoaster Inc. The total number of shares in Rollercoaster Inc. is 10^5, so Gina can not own more than 10^5 shares at any time. In Gina's daydream, the world is nice and simple: there are no fees for buying and selling stocks, stock prices change only once per day, and her trading does not influence the valuation of the stock.

Input

The first line of input contains an integer d ($1 \leq d \leq 365$), the number of days that Gina goes back in time in her daydream. Then follow d lines, the i'th of which contains an integer p_i ($1 \leq p_i \leq 500$) giving the price at which Gina can buy or sell stock in Rollercoaster Inc. on day i. Days are ordered from oldest to newest.

Output

Output the maximum possible amount of money Gina can have on the last day. Note that the answer may exceed 2^{32}.

Example

Input	Output
6 100 200 100 150 125 300	650

Problem E. Exponial

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

Everybody loves big numbers (if you do not, you might want to stop reading at this point). There are many ways of constructing really big numbers known to humankind, for instance:

- Exponentiation: $42^{2016} = \underbrace{42 \cdot 42 \cdot \ldots \cdot 42}_{2016\ times}$.
- Factorials: $2016! = 2016 \cdot 2015 \cdot \ldots \cdot 2 \cdot 1$.

Illustration of exponial(3) (not to scale), Picture by C.M. de Talleyrand-Périgord via Wikimedia Commons

In this problem we look at their lesser-known love-child the *exponial*, which is an operation defined for all positive integers n as

$$exponial(n) = n^{(n-1)^{(n-2)^{\cdots^{2^1}}}}$$

For example, $exponial(1) = 1$ and $exponial(5) = 5^{4^{3^{2^1}}} \approx 6.206 \cdot 10^{183230}$ which is already pretty big. Note that exponentiation is right-associative: $a^{b^c} = a^{(b^c)}$.

Since the exponials are really big, they can be a bit unwieldy to work with. Therefore we would like you to write a program which computes $exponial(n)$ *mod* m (the remainder of $exponial(n)$ when dividing by m).

Input

The input consists of two integers n ($1 \leq n \leq 10^9$) and m ($1 \leq m \leq 10^9$).

Output

Output a single integer, the value of $exponial(n)$ *mod* m.

Example

Input	Output
2 42	2
5 123456789	16317634
94 265	39

Problem F. Fleecing the Raffle

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

A tremendously exciting raffle is being held, with some tremendously exciting prizes being given out. All you have to do to have a chance of being a winner is to put a piece of paper with your name on it in the raffle box. The lucky winners of the p prizes are decided by drawing p names from the box. When a piece of paper with a name has been drawn it is not put back into the box – each person can win at most one prize.

Naturally, it is against the raffle rules to put your name in the box more than once. However, it is only cheating if you are actually caught, and since not even the raffle organizers want to spend time checking all the names in the box, the only way you can get caught is if your name ends up being drawn for more than one of the prizes. This means that cheating and placing your name more than once can sometimes increase your chances of winning a prize.

You know the number of names in the raffle box placed by other people, and the number of prizes that will be given out. By carefully choosing how many times to add your own name to the box, how large can you make your chances of winning a prize (i.e., the probability that your name is drawn exactly once)?

Input

The input consists of a single line containing two integers n and p ($2 \leq p \leq n \leq 10^6$), where n is the number of names in the raffle box excluding yours, and p is the number of prizes that will be given away.

Output

Output a single line containing the maximum possible probability of winning a prize, accurate up to an absolute error of 10^{-6}.

Example

Input	Output
3 2	0.6
23 5	0.45049857550

Problem G. Game Rank

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

The gaming company Sandstorm is developing an online two player game. You have been asked to implement the ranking system. All players have a rank determining their playing strength which gets updated after every game played. There are 25 regular ranks, and an extra rank, "Legend", above that. The ranks are numbered in decreasing order, 25 being the lowest rank, 1 the second highest rank, and Legend the highest rank.

Each rank has a certain number of "stars" that one needs to gain before advancing to the next rank. If a player wins a game, she gains a star. If before the game the player was on rank 6-25, and this was the third or more consecutive win, she gains an additional bonus star for that win. When she has all the stars for her rank (see list below) and gains another star, she will instead gain one rank and have one star on the new rank.

For instance, if before a winning game the player had all the stars on her current rank, she will after the game have gained one rank and have 1 or 2 stars (depending on whether she got a bonus star) on the new rank. If on the other hand she had all stars except one on a rank, and won a game that also gave her a bonus star, she would gain one rank and have 1 star on the new rank.

If a player on rank 1-20 loses a game, she loses a star. If a player has zero stars on a rank and loses a star, she will lose a rank and have all stars minus one on the rank below. However, one can never drop below rank 20 (losing a game at rank 20 with no stars will have no effect).

If a player reaches the Legend rank, she will stay legend no matter how many losses she incurs afterwards.

The number of stars on each rank are as follows:

- Rank 25-21: 2 stars
- Rank 20-16: 3 stars
- Rank 15-11: 4 stars
- Rank 10-1: 5 stars

A player starts at rank 25 with no stars. Given the match history of a player, what is her rank at the end of the sequence of matches?

Input

The input consists of a single line describing the sequence of matches. Each character corresponds to one game; '`W`' represents a win and '`L`' a loss. The length of the line is between 1 and 10 000 characters (inclusive).

Output

Output a single line containing a rank after having played the given sequence of games; either an integer between 1 and 25 or "`Legend`".

Example

Input	Output
WW	25
WWW	24
WWWW	23
WLWLWLWL	24
WWWWWWWWWLLWW	19
WWWWWWWWWLWWL	18

Problem H. Highest Tower

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

Oni loved to build tall towers of blocks. Her parents were not as amused though. They were on the verge of going crazy over that annoying loud noise whenever a tower fell to the ground, not to mention having to pick up blocks from the floor all the time. Oni's mother one day had an idea. Instead of building the tower out of physical blocks, why couldn't Oni construct a picture of a tower using two-dimensional rectangles that she montaged on a board on the wall? Oni's mother cut out rectangles of various sizes and colors, drew a horizontal line representing the ground at the bottom of the board, and explained the rules of the game to Oni: every rectangle must be placed immediately above another rectangle or the ground line. For every rectangle you can choose which of its two orientations to use. I.e., if a rectangle has sides of length s and t, you can either have a side of length s horizontally or a side of length t horizontally. You may place exactly one rectangle immediately above another one if its horizontal side is *strictly* smaller than the horizontal side of the rectangle beneath. Exactly one rectangle must be placed on the ground line. Now try to build as tall a tower as possible!

Oni's mother took extra care to make sure that it was indeed possible to use all rectangles in a tower in order not to discourage Oni. But of course Oni quickly lost interest anyway and returned to her physical blocks. After all, what is the point of building a tower if you cannot feel the suspense before the inevitable collapse? Her father on the other hand got interested by his wife's puzzle as he realized this is not a kids' game.

Input

The first line of input contains an integer n ($1 \leq n \leq 250\,000$), the number of rectangles. Then follow n lines, each containing two integers s and t ($1 \leq s \leq t \leq 10^9$ nm), the dimensions of a rectangle.

You may safely assume that there is a way to build a tower using all n rectangles.

Output

Output a single line containing the height in nm of the tallest possible tower using all the rectangles while having the horizontal side lengths strictly decreasing from bottom to top.

Example

Input	Output
3 50000 160000 50000 100000 50000 100000	200000

Problem I. Simple Median

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

The *Median* is a numeric value often used in descriptive statistics, and it is obtained as follows. You start by taking N values (the "data sample"), and sort them is ascending order. If the amount of values N is an odd number, you just take the data point right in the middle. For example, the set of numbers 4, 1, 3, when sorted becomes 1, 3, 4, and its median is the value in the middle: 3. When N is even, there is not a single value right in the middle. For example, the set 3, 2, 6, 1 when sorted becomes 1, 2, 3, 6 and the values from the middle will be 2 and 3. The median is then taken from the average of those two values. This means that in the previous example, the median would be $\frac{2+3}{2} = 2.5$.

For this problem, your task is to calculate the median from a data sample.

Input

The input is formed by several test cases. The first line of each test case contains the integer N, representing the amount of numbers from the sample. The next line contains the N integers from which you will calculate the median, $x_1\ x_2\ x_3\ \ldots\ x_i\ \ldots\ x_N$, separated by single spaces. The end of the input is signalled by a test case with $N = 0$, which must not be processed.

- $1 \leq N \leq 10^5$
- $1 \leq x_i \leq 10^9$

Output

For each test case given in the input, your program must print the median in a single line, rounded to the nearest tenth.

Example

Input	Output
1 5 2 5 4 3 5 5 4 0	5.0 4.5 5.0

Problem J. Jumbled Compass

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

Jonas is developing the JUxtaPhone and is tasked with animating the compass needle. The API is simple: the compass needle is currently in some direction (between 0 and 359 degrees, with north being 0, east being 90), and is being animated by giving the degrees to spin it. If the needle is pointing north, and you give the compass an input of 90, it will spin clockwise (positive numbers mean clockwise direction) to stop at east, whereas an input of −45 would spin it counterclockwise to stop at north west.

The compass gives the current direction the phone is pointing and Jonas' task is to animate the needle taking the *shortest path* from the current needle direction to the correct direction. Many `if`s, moduli, and even an arctan later, he is still not convinced his `minimumDistance` function is correct; he calls you on the phone.

Input

The first line of input contains an integer n_1 ($0 \leq n_1 \leq 359$), the current direction of the needle. The second line of input contains an integer n_2 ($0 \leq n_2 \leq 359$), the correct direction of the needle.

Output

Output the change in direction that would make the needle spin the shortest distance from n_1 to n_2. A positive change indicates spinning the needle clockwise, and a negative change indicates spinning the needle counter-clockwise. If the two input numbers are diametrically opposed, the needle should travel clockwise. I.e., in this case, output 180 rather than −180.

Example

Input	Output
315 45	90
180 270	90
45 270	-135

Problem K. Keeping the Dogs Apart

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

Despite the unfortunate incident last summer, which resulted in ten little puppies, you have been tasked with taking care of your neighbors' dogs again. Shadow and Lydia may be very cute mutts, but this year you have strict instructions to walk them one by one. However, you have other things to do during the summer than walking dogs! Like playing fetch and solving programming problems! It seems terribly inefficient to walk the dogs one at a time.

Shadow and Lydia have a particular walk they each prefer and know by heart. If you just let them out, they will follow their favorite walk, eventually ending up in their respective doghouses. Problem solved!

Sadly, you realize that if you just let both dogs out at the same time and let them do their walks on their own, they might get too close to each other. If they get too close, they will leave their favorite walk to "have some fun" and you are not sure you can find good homes for any more puppies. To ensure this does not happen, you need to calculate the minimum distance between the dogs when they are out walking on their own.

Both dogs start at the same time and keep exactly the same pace. Immediately after a dog arrives at its doghouse it stays inside and goes to sleep, so we no longer need to worry about the distance to the other dog, even though the other dog may still walk for a while longer. Note that a dog is still awake at the exact moment of entering its house and falls asleep immediately after entering.

Input

The first line of input consists of an integer n ($2 \leq n \leq 100\,000$), the number of points describing the walk of Shadow. The next n lines contain 2 integers each, giving the x and y coordinates of Shadow's walk. Two consecutive points in the walk always differ in at least one coordinate. All coordinates are non-negative and at most 10 000. Similarly, the next line contains an integer m ($2 \leq m \leq 100\,000$), the number of points describing the walk of Lydia. The next m lines describe its walk in the same format as for Shadow.

Output

Output the minimum distance between the two dogs during their walks. The numbers should be accurate to an absolute or relative error of at most 10^{-4}.

Example

Input	Output
2 0 0 10 0 2 30 0 15 0	10
5 10 0 10 8 2 8 2 0 10 0 9 0 8 4 8 4 12 0 12 0 8 4 8 4 12 0 12 0 8	1.414213562373