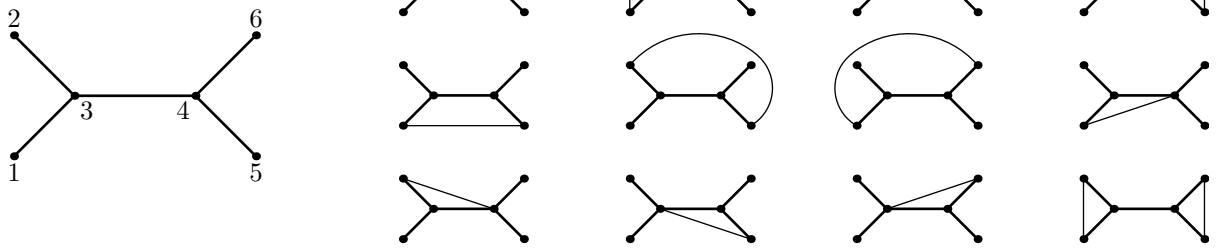
Problem A. Cactusability

Input file:	`cactus.in`
Output file:	`cactus.out`
Time limit:	1 second
Memory limit:	64 megabytes

Vertex cactus is a connected undirected graph in which every vertex lies on at most one simple cycle.

A tree can be converted to a vertex cactus by adding zero or more edges to it. There can be several ways to convert a tree to cactus. The number of ways to do so determines the *cactusability* of a tree.

For example, the cactusability of the tree on the left picture is 12. The twelve cactuses that it can be converted to are shown on the right.



Given a tree, you have to find its cactusability.

Input

The first line of the input file contains one integer number n — the number of vertices in a tree ($1 \le n \le 200$). The next $n - 1$ lines contain edges of a tree.

Output

Write to the output file a single integer number — the cactusability of the given tree.

Example

cactus.in	cactus.out
6 1 3 2 3 3 4 4 5 4 6	12

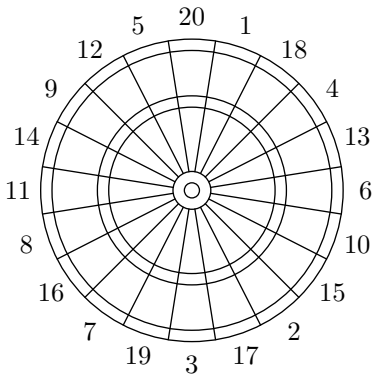
Problem B. Darts

Input file:	`darts.in`
Output file:	`darts.out`
Time limit:	1 second
Memory limit:	64 megabytes

The game of darts is played by two or more players. Players in turn throw three darts into the dartboard.

Initially each player has some number of points, usually 501. Each dart thrown at the dartboard *scores* some points. This number is subtracted from the player's points. The first one to reach exactly zero is the winner.

The dartboard looks as shown on a picture. There are 20 sectors arranged around the central circle, called the *bull eye*. This circle in turn consists of the inner bull eye (or simply the bull eye), and outer bull eye. Outer bull eye's score is 25, inner bull eye is double outer, which is 50. The outer sectors score the number of points they are marked with. There are also two circular areas going around the board. The outer circle area doubles the points of the sector, and the inner one triples them.



There are additional rules for the last round when the player is to decrease his score to zero. He can do it by throwing from one to three darts. First, the player must decrease it to exactly zero, going below zero is not allowed. Second, the last dart in a game must be double — that is, it must be thrown into the double part of some segment or into the bull eye (which is double outer bull eye).

For example, one correct way to finish the game from 50 points is by throwing darts to "`18`" and "`D16`". Finishing by playing "`D20`", "`10`", or "`20`", "`T10`" is illegal (the last throw in these cases is not double). Another correct way to win in this case is "`Bull`".

Given the number of points left, determine all the ways to finish the game correctly.

Input

The first line of the input file contains n — the number of points left ($1 \le n \le 200$).

Output

At the first line of the output file print k — the number of ways to correctly finish the game. After that print k lines, each must contain one correct way to finish the game. Use sector's value to denote it, preceed it with '`D`' for double area, '`T`' for triple area. Use "`25`" to denote the outer bull eye, "`Bull`" to denote the bull eye itself.

Examples

darts.in	darts.out
5	7 1 D1 D1 1 2 D1 1 D2 D1 1 D1 T1 D1 2 1 D1 3 D1

Problem C. Domino in Casino

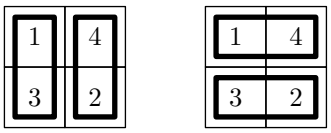
Input file:	`domino.in`
Output file:	`domino.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

Domino is well known as a game played by people at streets when they relax after a workday. So it was, until recently John Bigbuck introduced domino in his casino "*BUMP*" (Bring Us Money, Please).

Of course, ordinary domino games are not well suited for casino, so John had to introduce his own game. The game is played on a rectangular board of size $m \times n$. Each cell of the board contains some integer number.

The player has k domino tiles — rectangles 2×1. The player puts the tiles to the board without overlapping and his winning is calculated as the sum of products of numbers under each tile.

For example, there are 2 ways to put 2 tiles on a 2×2 board. For the board below, the better way to put the tiles is shown on the left — in this case the sum is $1 \times 3 + 4 \times 2 = 11$. If the player puts tiles to the board as shown on the right picture, the sum would be $1 \times 4 + 3 \times 2 = 10$, which is smaller.



Given the board, and the number of the tiles the player has, find what is the maximal sum he can get.

Input

The first line of the input file contains integer numbers m, n and k ($1 \le m \le 16$, $1 \le n \le 100$, $1 \le k \le 200$). The following m lines contain n integer numbers each and describe board. Numbers written on the board are non-negative and do not exceed 1000. It is guaranteed that it is possible to arrange all tiles on a board.

Output

Output one integer number — the maximal sum a player can get.

Examples

domino.in	domino.out
2 2 2 1 4 3 2	11

Problem D. Love Is. . .

Input file:	`love.in`
Output file:	`love.out`
Time limit:	1 second
Memory limit:	64 megabytes

Jessie likes Jane. She is very beautiful, he likes to watch at her. But Jessie likes Jill as well. She is also very beautiful. When she is sitting at the bench and reading a book, Jessie is ready to watch at her for years. Why not come and talk? There is a problem. Jessie is a dog.

But Jessie is not disappointed with the fact. Actually, he is quite used to it. But he really likes watching the girls. Unfortunately, there are many houses and other obstacles that try to break Jessie's plans. So sometimes it is impossible to watch at both girls simultaneously. Sometimes it is possible, but it is always hard to find the apropriate point. Jessie asks you, his master, help him.

Given the location of the obstacles and the points where the girls are, find the point from which both girls are visible, or find out that there is no such point. Of course, Jessie cannot get inside obstacles.

Input

The first line of the input file contains two integer numbers x_1 and y_1 — the coordinates of Jane. The second line contains x_2 and y_2 — the coordinates of Jill. The third line of the input file contains n — the number of obstacles ($0 \le n \le 10$).

All obstacles are rectangles with sides parallel to coordinate axis. The following n lines contain four integer numbers each $x_{i,1}$, $y_{i,1}$, $x_{i,2}$ and $y_{i,2}$ — the coordinates of the bottom-left and top-right corners of the obstacles. All coordinates do not exceed 100 by their absolute value. Obstacles do not intersect but may touch each other. If two obstacles touch each other by corners, or sides there is no hole. In the other case Jessie can look so that his line of sight touches the corner or goes along the side of the obstacle.

No girl is inside or on the border of any obstacle. Girls' positions do not coincide.

Output

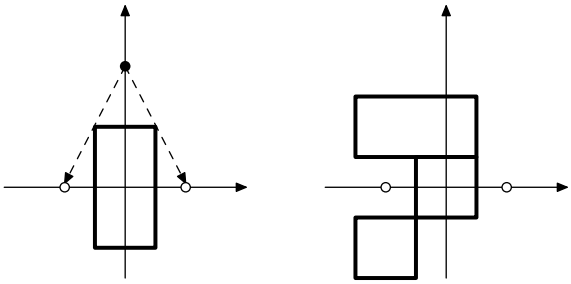
If there is a point from which Jessie can see both girls, print "`YES`" at the first line of the output file. In this case the second line of the output file must contain two real numbers — the coordinates of the point from which Jessie must watch. The point must not be inside any obstacle, but may be on its border. Jessie must not be at the point that belongs simultaneously to two corners of the buildings that do not have a common side.

The coordinates must be accurate up to at least 10^{-6}.

If there is no such point, print "`NO`" at the first line of the output file.

Examples

love.in	love.out
2 0 -2 0 1 -1 -2 1 2	YES 0.0 4.0
2 0 -2 0 3 -1 -1 1 1 -3 -3 -1 -1 -3 1 1 3	NO



Problem E. Set Partitions

Input file:	`partitions.in`
Output file:	`partitions.out`
Time limit:	1 second
Memory limit:	64 megabytes

Let us consider the set of n first integer numbers: $N_n = \{1, 2, \ldots, n\}$. The *partition* is the representation of this set as a union of one or more non-empty disjoint sets. The examples of the partitions for $n = 5$ are:

$$\{1,2,3,4,5\} = \{1,2,3\} \cup \{4,5\}$$
$$\{1,2,3,4,5\} = \{1,3,5\} \cup \{2,4\}$$
$$\{1,2,3,4,5\} = \{1,2,3,4,5\}$$
$$\{1,2,3,4,5\} = \{1\} \cup \{2\} \cup \{3\} \cup \{4\} \cup \{5\}$$

There are totally 52 partitions of the set N_5. Note that we do not distinguish between partitions that only differ by the order of the united sets.

We can order N_n partitions in the *lexicographical order*.

To define this order we first introduce lexicographical order on the subsets of N_n. We say that the set $A \subset N_n$ is lexicographically smaller than the set $B \subset N_n$ and write $A < B$ if one of the following is true:

- exists i such that $i \in A$, $i \notin B$, for all $j < i$: $j \in A$ iff $j \in B$, and exists $k > i$ such that $k \in B$;
- $A \subset B$ and $i < j$ for all $i \in A$ and $j \in B \setminus A$.

It is clear that the relation introduced is the total order on all subsets of N_n.

In other words, A is lexicographically less than B as a sequence if written in the increasing order.

Now we define the *canonical representation* of the partition as the representation in which the united sets are ordered lexicographically.

The partitions are ordered lexicographically in the following way. The partition $N_n = A_1 \cup A_2 \cup \ldots \cup A_k$ is lexicographically smaller than the partion $N_n = B_1 \cup B_2 \cup \ldots \cup B_l$ if there is such i that $A_1 = B_1$, $A_2 = B_2$, $\ldots$, $A_{i-1} = B_{i-1}$ and $A_i < B_i$.

Given the partition of N_n you have to find the next partition in the lexicographical order.

Input

The input file contains several tests cases. Each test case is the the canonical representation of the partition. The first line of the test case contains n and k — the number of elements in the partitioned set and the number of parts in a partition ($1 \le n \le 200$). The following k lines contain elements of the partition. Elements of each set are sorted in the increasing order.

Test cases are separated from each other with blank lines. The last line of the output file contains two zeroes. This case must not be processed.

The sum of n in all testcases of the input file doesn't exceed 2000.

Output

For each test case output the next partition in the lexicographical order. If the partition in the input file is the last one, output the first in lexicographical order partition. Use the same format as the input file. Separate output for test cases by a blank line.

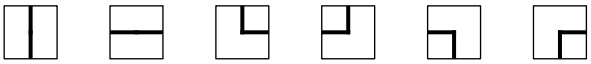
Examples

partitions.in	partitions.out
5 2 1 2 3 4 5 5 2 1 3 5 2 4 5 1 1 2 3 4 5 5 5 1 2 3 4 5 0 0	5 2 1 2 3 4 5 5 4 1 4 2 3 5 5 2 1 2 3 5 4 5 4 1 2 3 4 5

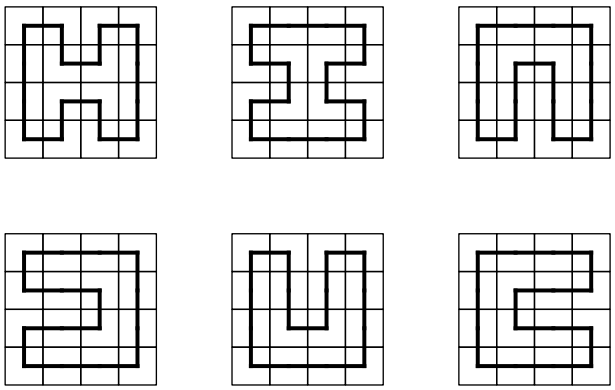
Problem F. Pipe Layout

Input file:	`pipe.in`
Output file:	`pipe.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

The city is building a centralized heating system in some of the city districts. City district occupies rectangular area and consists of square blocks arranged in a grid. Centralized heating system is a closed circuit of pipes that will be used to run hot water through every block in the district. The city council is considering different layouts of pipes for each district. In order to minimize the total length of pipes but still provide hot water to every block in the district, exactly one pipe must run through every block. A pipe in every block must be connected to the pipes in two neighboring blocks. So, there are at most six possible pipe configurations in every block:



In order to plan their planning activities, the city council wants to know the number of different possible pipe layouts for a given city district. For example, there are exactly 6 different pipe layouts for a district with 16 blocks arranged into 4 by 4 grid:



Input

The input file contains two integer numbers r ($r > 1$) and c ($c > 1$). They specify, correspondingly, the number of rows and columns of blocks in the district. The total number of blocks in a district does not exceed 100 ($r \times c \leq 100$).

Output

Output the number of different possible pipe layouts for this district.

Examples

pipe.in	pipe.out
4 4	6
5 7	0
2 8	1
12 8	102283239429

Problem G. Word Square

Input file:	square.in
Output file:	square.out
Time limit:	1 second
Memory limit:	64 megabytes

Some sets of n words of length n have a nice property — it is possible to arrange them in the $n \times n$ grid so that this set of words is read both vertically and horizontally.

An example of such set is { "DATE", "FIND", "IDEA", "NEXT" }. You can arrange them in the following way:

F	I	N	D
I	D	E	A
N	E	X	T
D	A	T	E

Note that you can find each word exactly once as a horizonatal word, and as a vertical word. Such squares are called *word squares*, the largest known word square is 10×10.

Another example of the 4×4 word square is:

C	R	A	B
R	A	R	E
A	R	T	S
B	E	S	T

You are given $2n$ words such that it is possible to construct two different $n \times n$ word squares from them. You task is to divide the words into two groups of n words and construct the words square from each group.

You are guaranteed that all words are the English words (however, some of them are really rare ones, human names, or some extremely special terms).

Input

The first line of the input file contains n ($2 \le n \le 10$). The following $2n$ lines contain n-letter word each, words consist of capital English letters.

Output

Output two $n \times n$ word squares constructed from the given words. Separate squares by a blank line.

Examples

square.in	square.out
4 ARTS BEST CRAB DATE FIND IDEA NEXT RARE	FIND IDEA NEXT DATE CRAB RARE ARTS BEST

Problem H. Subword

Input file:	`subword.in`
Output file:	`subword.out`
Time limit:	1 second
Memory limit:	64 megabytes

Let us introduce the free group with two non-commuting generators.

We will denote generators as a and b and their reverses as A and B respectively. The group can be described as a set of words consisting of a, b, A and B, the operation of concatentation, and the generating relations $aA = Aa = Bb = bB = \varepsilon$.

In other words, we say that two words are equivalent if one can be transformed to another by successively performing the following operations:

- remove a subword equal to Aa, aA, Bb or bB from any position;
- add a subword equal to Aa, aA, Bb or bB to any position.

For example, words $abAaBBabbA$ and $aAaBabaAbA$ are equivalent:

$$abAaBBabbA \to abBBabbA \to aBabbA \to aAaBabbA \to aAaBabaAbA,$$

but words $abAB$ and $baBA$ are not.

Equivalence classes of words are exactly the elements of the group.

It is interesting to note that for each word α and each word β there is such word γ equivalent to α that contains β as a subword. Your task in this problem is to find the shortest such word.

Input

The first line of the input file contains α. The second line contains β. Both words are not empty and contain at most 2000 characters each.

Output

Output one word — the shortest word equivalent to α that contains β as a subword. If there are several solutions, print any one.

Examples

subword.in	subword.out
abAaBBabbA AaBaba	aAaBabaAbA

Problem I. HTML Table

Input file:	`table.in`
Output file:	`table.out`
Time limit:	1 second
Memory limit:	64 megabytes

The company *Kozilla* is developing the new browser *Waterrat*. Your task is to develop the part of the browser responsible for rendering tables.

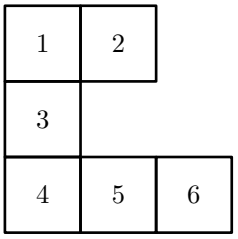
In this problem we will consider the simplified version of HTML. We will only consider the part of HTML that describes tables. HTML document is the sequence of opening and closing *tags*. The contents of HTML document is located between the tags.

The opening tag is specified as `<`*`tag-name`* *attributes*`>`, the closing tag — `</`*`tag-name`*`>`. Here attributes mean the sequence of *`name`*`="`*`value`*`"` descriptions. Attributes are used to specify various properties of web-page elements. Table is a set of cells ordered by rows and columns. Tags `<table>` and `</table>` are used to specify a table. The description of the table is between the tags.

The description of the table is the sequence of the descriptions of table rows. Row description is located between `<tr>` and `</tr>` tags and consists of a sequence of cell descriptions. Cell description consists of a `<td>` — `</td>` tag pair, the contents of the cell is located between these tags.

Let us consider an example of a table and the way it is displayed in a browser.

```
<table>
<TR><td>1</td><td>2</td></TR>
<TR><td>3</td></TR>
<TR><td>4</td><td>5</td><td>6</td></TR>
</table>
```



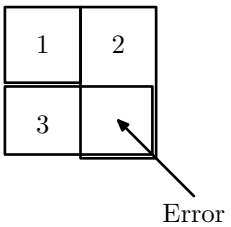
You must have noticed that some cells may be absent, and tag names are not case sensitive.

To draw more complicated tables, one can specify a table cell that would occupy more than one row and/or column. To do so, one must add attributes `rowspan="`R`"` and/or `colspan="`C`"` to the `<td>` tag. Such cells expand down to span R rows and to the right to span C columns.

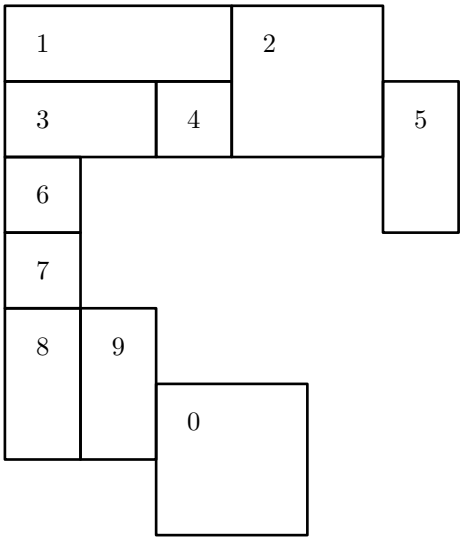
The table is rendered row after row. When creating a row, and the new cell description is considered, the first free position in the row is selected. Let the cell being rendered occupy C columns. If there are C successive free positions in a row, starting from the selected one, the cell occupies these positions (and spans down to the required number of rows). In the other case the table description is considered erroneous. The behaivour of the browser in this case is not specified, your routine must report error.

Let us consider two more examples.

```
<table>
<tr>
  <td>1</td>
  <td rowspan="2">2</td>
</tr>
<tr>
  <td colspan="2">3</td>
</tr>
</table>
```



```
<table>
<tr>
  <td colspan="3">1</td>
  <td rowspan="2" colspan="2">2</td>
</tr>
<tr>
  <td colspan="2">3</td>
  <td>4</td>
  <td rowspan="2">5</td>
</tr>
<tr>
  <td>6</td>
</tr>
<tr>
  <td>7</td>
</tr>
<tr>
  <td rowspan="2">8</td>
  <td rowspan="2">9</td>
</tr>
<tr>
  <td colspan="2" rowspan="2">0</td>
</tr>
</table>
```



Given a table description, you must print the outlines of the table to the output file. You must ignore the contents of the cells, all cells must contain only spaces.

Input

The input file contains a web-page fragment that describes the table. It is guaranteed that the description is correct and describes exactly one table. The input file contains only `<table>`, `</table>`, `<tr>`, `</tr>`, `<td>` and `</td>` tags. Only `<td>` tags have attributes, only `rowspan` and `colspan` attributes are used. Tag and attribute names are case insensitive. The values of `rowspan` and `colspan` attributes are positive integer numbers not exceeding 10.

The table describes at least one row, each row description contains the description of at least one cell.

The contents of the cell is specified between `<td>` and `</td>` tags and may contain arbitrary text, containing characters with ASCII codes from 33 to 126 except '`<`', as well as spaces and line breaks.

Spaces and line breaks may occur between any tags; inside tags between the tag name and the attribute name, between the tag name and the '`>`' character, between the attribute name and the '`=`' character, between the '`=`' character and the '`"`' character before the attribute value, between the '`"`' character after the attribute value and the following attribute name, between the '`"`' character after the last attribute value and the '`>`' character. They must be ignored.

It is guaranteed that the table described contains at most 100 cells, the size of the input file does not exceed 10 kilobytes.

Output

Print the outline of the table to the output file.

Use the following characters:

'`+`' (plus) — to display line intersections;
'`|`' (vertical line, ASCII code 124) — to display vertical lines;
'`-`' (minus) — to display horizontal lines;
' ' (space) — in all other cases;

The cell that occupies m rows and n columns must be displayed as $2m - 1$ rows of $2n - 1$ spaces.

If the table is erroneous, print "ERROR" to the output file.

Examples

table.in	table.out
<table> <TR><td>1</td><td>2</td></TR> <TR><td>3</td></TR> <TR><td>4</td><td>5</td><td>6</td></TR> </table>	+-+-+ | | | +-+-+ | | +-+-+-+ | | | | +-+-+-+
<table> <tr> <td>1</td> <td rowspan="2">2</td> </tr> <tr> <td colspan="2">3</td> </tr> </table>	ERROR
<table> <tr> <td colspan="3">1</td> <td rowspan="2" colspan="2">2</td> </tr> <tr> <td colspan="2">3</td> <td>4</td> <td rowspan="2">5</td> </tr> <tr> <td>6</td> </tr> <tr> <td>7</td> </tr> <tr> <td rowspan="2">8</td> <td rowspan="2">9</td> </tr> <tr> <td colspan="2" rowspan="2">0</td> </tr> </table>	+-----+---+ | | | +---+-+ +-+ | | | | | +-+-+-+---+ | | | | | +-+ +-+ | | +-+-+ | | | | | +---+ | | | | +-+-+ | | | +---+

Problem J. Wheel of Fortune

Input file:	`wheel.in`
Output file:	`wheel.out`
Time limit:	1 second
Memory limit:	64 megabytes

The popular show "*Wheel of Fortune*" attracts people from the whole country to their TV screens at the Friday night. The show proceeds as follows.

There are n persons taking part in a show. The goal of the show is to guess the secret word. Initially only the number of letters in a word l is known, the letters are shown at the special tableau as the black boxes.

The player who makes the turn chooses some letter. If this letter exists in the secret word, all of its occurences are open and the player makes the next turn. If there is no such letter the turn moves over to the next player. After the last player the turn gets back to the first one. The player who guesses the last letter is the winner of the game.

For example, let the secret word be "`CONTEST`". Initially the players only see "`-------`". If the first player says '`E`', it is opened, and the players see "`----E--`". The first player makes turn again, let him, for example, say '`A`'. There is no such letter in the secret word, so the turn passes on to the next player. If he says '`T`', its occurences are opened and the players see "`---TE-T`", and so on.

Paul's friend is going to take a part in the game. Paul would like to know what are the chances of his friend. For each player Paul has learned his intellectual potential q_t. The probability that the player t would guess the letter correctly, if there are i letters still not suggested, j different letters of the word are still unknown, and k unopened boxes on the tableau, is the following:

$$p_{t,i,j,k} = \left(1 - q_t^{\frac{1}{k-j+1}}\right)\frac{j}{i} + q_t^{\frac{1}{k-j+1}}\left(1 - \frac{1}{i}\right)^{i-j}.$$

Here we consider $0^0 = 1$.

Each of the unknown letters has the equal probability to be guessed.

Given n, the Paul's friend's position among the players r, the secret word, and the intellectual potentials of the players, help Paul to detect what is the probability that his friend would win in the game.

Input

The first line of the input file contains n and r ($2 \le n \le 10$, $1 \le r \le n$). The second line contains the secret word. It consists of capital letters of the English alphabet. Its length does not exceed 12. The third line of the input file contains n real numbers — the intellectual potentials of the players ($0 \le q_t \le 0.99$).

Output

Output one number — the probability that Paul's friend would win the game. Your answer must be accurate up to 10^{-8}.

Examples

wheel.in	wheel.out
3 1 CONTEST 0.7 0.2 0.1	0.4648222937