Problem A. Divide the Cash

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

The UCF Programming Team coaches schedule practices regularly in fall and spring (by the way, all UCF students are welcome to the practices). During summer, the majority of the team members are gone but the coaches know how to make sure the students don't get "rusty". The coaches usually give prize money and reward the team members based on how many problems they solve during summer. For example, let's assume the coaches have $\$1,000$ in prize money and there are three students. Let's also assume the three students solve, respectively, 5, 8 and 7 problems, i.e., a total of 20 problems. So, the reward for one problem will be \$50 (\$1000/20) and the three team members will receive, respectively, \$250, \$400 and \$350.

Given the total prize money and the number of problems each team member has solved, compute the reward for each student.

Input

The first input line contains two integers: n $(1 \leq n \leq 30)$, indicating the number of team members and d $(1 \leq d \leq 30,000)$, indicating the dollar amount to be divided among the students. Each of the next n input lines contains an integer (between 1 and 300, inclusive) indicating the number of problems a team member has solved. Assume that the input values are such that the reward for one problem will be an integer number of dollars.

Output

Print the pay, in dollars, for each team member (in the same order as they appear in the input).

Example

Input	Output

3 1000
5
8
7
	250
	400
	350

1 500 10	500

Problem B. Sort by Frequency

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

We all have heard the expression "more is better" so, in this problem, we are going to give higher precedence to the letter occurring the most.

Given a string containing only lowercase letters, you are to sort the letters in the string such that the letter occurring the most appears first, then the letter occurring the second most, etc. If two or more letters occur the same number of times, they should appear in alphabetical order in the output.

Input

The input consists of a single string, appearing on a line by itself, starting in column 1 and not exceeding column 70. The input will contain only lowercase letters (at least one letter).

Output

Print the sorted version of the input string, all on one line with no spaces.

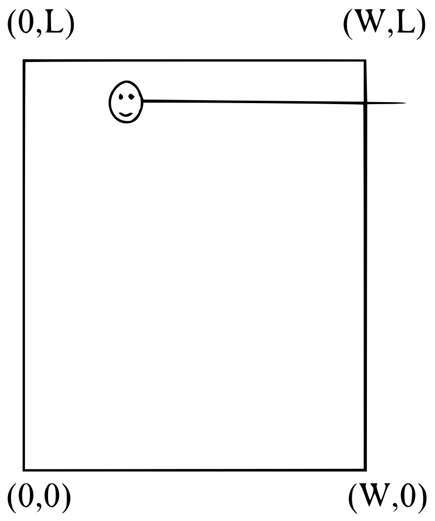
Example

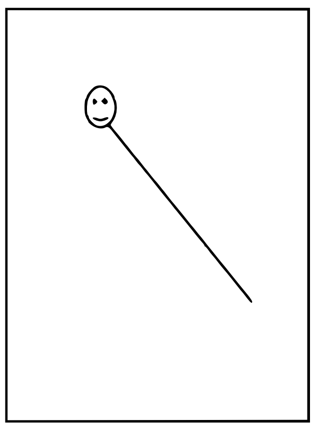
Input	Output
dcbbdabb	bbbbddac
programming	ggmmrrainop

Problem C. Fitting on the Bed

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

Arup's daughter, Anya, doesn't like sleeping in a bed in the usual way. In particular, she is willing to put her head anywhere, and furthermore, she's willing to lie in any direction. For the purposes of this problem, Anya, who is quite skinny, as she's only at the fifth percentile in weight, will be modeled as a line segment with her head being one of the end points of the segment. The bed will be modeled as a rectangle with the bottom left corner with coordinates $(0, 0)$, the top left corner with coordinates $(0, L)$, where L is the length of the bed, and the top right corner with coordinates (W, L), where W is the width of the bed. In the left diagram below, Anya's head is near the top left corner and she's sleeping completely sideways, off the bed - this corresponds to the first sample case. In the right diagram below, her head starts at a point a bit below where it starts in the first diagram, but she's sleeping at a diagonal, which allows her to fit on the bed!





Given the size of the bed, the location of Anya's head, her height, and the angle in which her body is in relation to her head (here we use 0 degrees to indicate that her body is going straight to the right of her head and 90 degrees to indicate that her body is going above her head), determine if Anya is fully fitting within the bed or not. (Note: a normal sleeping position would be having your head near $(W/2, L)$ with your body in the direction of 270 degrees.)

Input

The first line of input contains three positive integers: L ($L \leq 500$), the length of the bed in centimeters, W ($W \leq 500$), the width of the bed in centimeters, and H ($H \leq 200$), Anya's height in centimeters. The second line of input contains three non-negative integers: x ($x \leq W$), the number of centimeters from the left side of the bed Anya's head is, y ($y \leq L$), the number of centimeters from the bottom of the bed Anya's head is, and a ($a \leq 359$), the angle in degrees Anya's body is facing in relation to her head

Output

On a line by itself, output `YES` if Anya completely fits on the bed, or `NO`, otherwise. Note: for all cases where Anya doesn't fit on the bed, at least 1% of her body will be off the bed. She's considered on the bed if part or all of her touches edges of the bed but none of her body extends off the bed.

Example

Input	Output
200 150 115 40 190 0	NO
200 150 115 40 150 315	YES

Explanation

In the first sample case, Anya's head is near the top of the bed 40 centimeters from the left end of the bed, but her body is veering completely to the right. Since there is only 110 cm to the right and Anya is 115 cm tall, 5 cm of her is off the bed. In the second sample case, Anya's head is 40 cm below where her head was for the first sample case, but she's sleeping in a diagonal in the direction of where the hour hand of a clock would be at 4:30. In this configuration, she has enough room to fit on the bed, since she can go up to 110 cm to the right and 150 cm down.

Problem D. Candy Land

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

A popular board game among young children is Candy Land. The game board has several squares in a row and each square is either a color or a special square. Several players can play and on a player's turn, she turns over a card from the top of a deck, which is face down. Each of these cards will have either a square with a color, two squares with the same color, or a picture of a special square. If the card has a square with a color, the player advances (from their current location) to the next square on the board with that color. If the card has two squares with the same color, the player advances to the second square on the board with that color past their current location. Finally, if the card has a special square pictured, the player moves directly to that special square (this move could move the player ahead or behind where they currently are; assume all special squares are unique, i.e., there is exactly one occurrence of each special square on the board). Note that it is possible for multiple players to be in the same square at the same time.

The last square of the board is multicolored (all colors) and whoever reaches that square first wins. Also, if a player pulls a card with two squares with the same color, but only the multicolored square at the end has that color on it, the player wins.

Since young children don't know how to shuffle cards, when they play the game, the cards are initially set in a particular order. After a player picks up a card and follows its move, she places that card, face down, at the bottom of the deck. Due to the lack of shuffling, what the children playing don't realize is that the game is 100% deterministic!

Given the layout of all the squares on a version of Candy Land, the number of players in the game, and the order of the cards in the deck, determine which player will win the game.

Input

The first line of input contains two positive integers: n ($2 \leq n \leq 200$), the number of squares on the Candy Land board and p ($2 \leq p \leq 6$), the number of players in the game. The following $n-1$ lines of input will each contain a single string of uppercase letters, indicating the contents of each square on the board, in order, except the last square. The last square is all colors and indicates where a player lands if there are no further squares of that color to land on. Note that players start directly prior to the first square of the board. Each of these strings will contain at most 20 uppercase letters. Each special square will start with the prefix "SPECIAL" followed by at least one uppercase letter (assume none of the colors will contain the sub-string "SPECIAL").

The input line following the board definition will contain a single positive integer, c ($c \leq 500$), representing the number of cards in the deck for the game. The following c lines of input contain the contents of the deck, one card per line, with the top card of the deck first. Each card is described with a number (1, 2 or 3) followed by a space followed by a string of uppercase letters. The number 1 indicates a card with a single square. The number 2 indicates a card with two squares. Both of these types of cards are followed by a string guaranteed to be a color that appears on the board. The number 3 indicates a special square. This is followed by a string guaranteed to be a valid special square on the board.

Output

On a line by itself, output the 1-based number of the player who wins the game. Note: it is guaranteed that the input game finishes in fewer than 10,000 turns, though it's possible that if play were infinitely extended, some of the players would never finish.

Example

Input	Output

`10 2`
`RED`
`BLUE`
`SPECIALCANE`
`GREEN`
`RED`
`BLUE`
`BLUE`
`GREEN`
`RED`
`4`
`1 RED`
`2 BLUE`
`3 SPECIALCANE`
`2 GREEN`
	`2`

`3 5`
`RED`
`SPECIALLOLLIPOP`
`2`
`3 SPECIALLOLLIPOP`
`1 RED`
	`1`

Explanation

In the first sample game, the first player moves to the first square, a red square. Then, the second player moves to the second blue square, which is square number 6. Then, the first player moves to the cane square, which is square number 3. Finally, the second player gets to move 2 green squares forward. Since there is only one green square (#8) in front of player 2, she wins the game by getting to square number 10. Had there been another turn in the game, the first player would have gotten the card with 1 red square. In the second sample game, player #1 wins on her second turn. She pulls the special card followed by the red card.

Problem E. Give-a-Gnocchi

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

Your favorite Italian restaurant is having a promotion. The restaurant is giving away "free" gnocchi soups. The only cost is to provide them with a prime number. But the restaurant does not want to go out of business; they will only accept a prime number if it is not already in their database. Once a prime number is accepted, they add it to their database. As it turns out, every time you have tried to give them a prime number, no matter how large, they already have it in their database, so you end up having to pay for your meal.

Your friend, Euler, noticed something odd about the verification program. Euler gave the restaurant the number $24,990,001$, and they accepted it. Euler realized this value was not prime after he checked it at home, so he did some analysis to determine what happened. It turns out the restaurant's database accepts any number (even a composite number!) that is not divisible by a prime lower than or equal to some value $\boldsymbol{n}$; which means you can easily get your gnocchi soup (albeit a little dishonestly) by finding a number (even a composite) the restaurant will accept. Basically, on the first day, you could give the first composite number that is not divisible by a prime lower than or equal to $\boldsymbol{n}$; on the second day, you would give the next composite number that is not divisible by a prime lower than or equal to $\boldsymbol{n}$, etc.

Since the restaurant keeps track of the numbers given to them, it is difficult to find what number to give to the restaurant next. Your other friend, Fermat, suggests that you write a program that will take in a value $\boldsymbol{n}$ and a day $\boldsymbol{k}$, and it will output the k^{th} (1-indexed) composite number that is not divisible by a prime less than **n**.

Input

The input consists of a single line providing two positive integers: n ($n \leq 1000$), representing the highest possible value for checking for primality, and k ($k \leq 1000$), indicating the index of the desired composite number.

Output

Print the k^{th} (1-indexed) composite number satisfying the given constraints.

Example

Input	Output
`10 3`	`169`
`1 1`	`4`
`19 7`	`943`

Explanation

In the first sample case, the smallest composite numbers not divisible by a value less than or equal to 10 are 121, 132, 169, 189, 221. On the first day we could give 121. The second day we would use 132. On the third day (when $k = 3$) we would use 169.

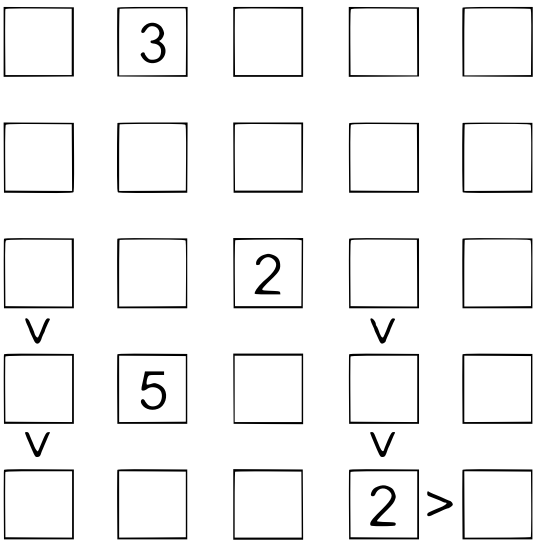
For the second sample case, the answer is 4, since it is both the 1^{st} composite number and the first number not divisible by any prime less than or equal to 1.

For the last case, the smallest composite numbers not divisible by a prime less than or equal to 19 include 529, 667, 713, 841, 851, 899, 943, 961, and 989. The 7^{st} value in the list is 943, which will be the value given on the 7^{st} day.

Problem F. More or Less

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

You thought it would never happen, the Sudoku craze is finally over! People are overwhelmingly bored of Sudoku puzzles. To take advantage of their fatigue, you've decided to make up a new puzzle, called "More or Less", so that you can become rich! Naturally, so that the old Sudoku solvers won't be lost, your rules must be of a similar format to Sudoku, but nonetheless be different. You've decided on an $n \times n$ board, where all valid solutions must have each integer 1 through n, inclusive, appear on each row and each column. There is no box or diagonal rules. Naturally, some numbers out of the n^2 squares are given and the puzzle is to fill the rest out. In addition to some numbers being given, some less than and greater than signs are placed between pairs of adjacent squares. Consider the puzzle shown below:



In this puzzle, we are given that the entry in the last row and column is less than the entry in row 5, column 4, which is 2. Since we must use the integers 1 through 5 only, it follows that the entry in the last row and column must be 1. Similarly in this puzzle, we see that the entry in row 4, column 4 must be greater than 2 and the entry in row 3 column 4 has to be bigger than the entry in row 4, column 4.

In order for you to be able to mass market this puzzle, you have to write a program that automatically solves arbitrary puzzles, since you don't have time to do such nonsense by hand.

Given a More or Less puzzle which has precisely one unique solution, determine the correct values that must go in each of the blank squares.

Input

The first line of input contains a positive integer, n ($n \leq 7$), representing the number of rows and columns in the More or Less puzzle. The puzzle will follow on the next $2n-1$ lines. The odd line numbers $(1, 3, ...)$ will contain the rows of the puzzle while the even lines will contain information about relationships between vertically oriented adjacent cells. On each of the odd lines, the odd numbered characters will represent the contents of each box of the puzzle, with empty squares represented with a '`-`' (hyphen character) and filled in numbers with the appropriate digits in between 1 and n, inclusive. On each of the odd lines, the even numbered characters will either be '`.`' (period), '`>`' (greater than) or '`<`' (less than). A period denotes no relationship between the two squares it is in between while the inequality signs denote their natural relationship.

On the even numbered lines, the odd numbered columns will either contain '`.`' (period), 'v' (lowercase v) or ‘^’ (caret symbol). A period denotes no relationship between the two squares it is in between while the ‘v’ denotes that the value in the square above it is greater than the value in the square below it and the ‘^’ denotes that the value in the square above it is less than the value in the square below it. All of the even column characters on the even numbered rows will contain ‘.’ only.

It is guaranteed that each puzzle will have a unique solution, which means that all of the information on the given input board will be consistent with itself, and there will be exactly one completed grid that will satisfy all of the given constraints.

Output

Output n rows of exactly n digits representing the solution to the puzzle. Do not output spaces or any other characters.

Example

Input	Output
5	23415
-.3.-.-.-	12354
....	51243
-.-.-.-.-	45132
....	34521
-.-.2.-.-	
v.....v..	
-.5.-.-.-	
v.....v..	
-.-.-.2>-	
4	4123
-.-.-.-	3241
..^....	2314
-.-.->-	1432
....	
-<-.-.-	
v.^....	
-.-.-.2	

Problem G. Cooperative Escape

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

Bonnie and Clyde have noticed that parallel processing improves throughput so, instead of robbing one bank together, they've decided to rob two different banks simultaneously (each robbing one). They both succeeded and now they need to run to their get-away 1934 Ford Model (730 Deluxe Sedan). There are complications, of course, or the movie won't be exciting!

The city is in the shape of a matrix with cells. Bonnie and Clyde are in two different cells, and the get-away car is in a different cell as well. Bonnie and Clyde can move in four directions: north, south, east, and west (the border cells of course have fewer move options since they cannot move across the outer walls of the city). Some cells are also blocked, i.e., neither one can move into it. To put more adventure in the movie, it is also the case that once Bonnie or Clyde moves into a cell, then neither one can move into that cell anymore, i.e., a cell is used at most once. Bonnie and Clyde cannot move into each other's starting position either.

Given the city map (in the form of a matrix), the location for Bonnie, Clyde, and car, you are to compute the minimum total number of moves needed for both Bonnie and Clyde to get to the car, i.e., the total number of moves made by Bonnie plus the total number of moves made by Clyde.

Input

The first input line contains two integers, r and c ($2 \leq r, c \leq 30$), providing the number of rows and columns for the matrix. Each of the next r lines contains c characters, starting in column 1; this is the input matrix. There will be exactly one "B" (Bonnie's starting location), one "C" (Clyde's starting location), and one "F" (where the get-away Ford car is) in the input maze. The remaining cells will be "." (empty cell – one can move into) or "x" (blocked – one cannot move into).

Output

Print the minimum total number of moves needed for both Bonnie and Clyde to get to the car. If they cannot get to the car, print -1. Note that when one reaches the car, he/she stops moving.

Example

Input	Output

`6 5`
`.....`
`...C.`
`.B.x.`
`.....`
`...F.`
`.....`
	`9`

`3 7`
`.....F.`
`xxx.xxx`
`C.....B`
	`-1`

`7 5`
`C....`
`...B.`
`xx.x.`
`.....`
`.F...`
`.....`
`.....`
	`14`

Explanation

In the first sample case, Bonnie can reach the car with 4 moves and Clyde can reach the car with 5 moves, for a total of 9 moves.

Problem H. Assessing Genomes

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

The world is at the brink of extinction. A mutated virus threatens to destroy all living organisms. As a last hope, a team of super-smart scientists, including – of course – you, is currently working on an antivirus. Unfortunately, your team is unable to analyse the DNA in time. They sequenced n parts of the virus' DNA and need to match them with n available strands for antiviruses. As the algorithms expert, you need to implement a specialised procedure to solve this problem. Your approach needs to be fast – there is not much time left!

You first need to determine the *repetition score* of each DNA sequence. The repetition score of a sequence s is equal to the length of the shortest sequence u such that s is equal to the k-fold repetition of u, for some positive integer k. For instance, `ATGATG` has a repetition score of 3, since it can be produced by repeating `ATG` two times. On the other hand, `ATATA` has a repetition score of 5, as it cannot be produced from any proper substring.

Once you obtained the scores of all sequences, you need to match the n antivirus sequences with the n virus sequences in a way that minimises the damage caused by the virus. When two sequences are matched, the damage caused by the virus is equal to the squared difference between the two repetition scores. For instance, matching the antivirus sequence `ATGATG` with the virus sequence `ATATA` causes $(3-5)^2 = 4$ units of damage.

If you match the DNA sequences optimally, what is the minimal total damage caused by the virus, taken as a sum over all matched pairs?

Input

The input consists of:

- A line with an integer n ($1 \leq n \leq 50$), the number of DNA sequences of the virus and antivirus each.
- n lines, each with a virus DNA sequence.
- n lines, each with an antivirus DNA sequence.

Each DNA sequence is a non-empty string with a length of at most 250 and consists of lowercase letters `a-z` and uppercase letters `A-Z`.

Output

Output one integer, the minimal total damage.

Example

Input	Output
2 TTTTTT TATG TATATA AAAGAAAG	1
3 abcdef aaa bab AbAb xyzxyz X	10

Problem I. Floating-Point Unrounding

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

Juliana would like to extrapolate, as accurately as possible, a geometric sequence with terms that have all been rounded to D significant digits.

A geometric sequence is an ordered sequence of real numbers of the form a_0, a_1, a_2, a_3, …, a_{n-1}, where the terms (i.e., the numbers in the sequence) are related by powers of a constant factor r, specifically, $a_1 = a_0 \cdot r$, $a_2 = a_0 \cdot r^2$, $a_3 = a_0 \cdot r^3$, …, and $a_{n-1} = a_0 \cdot r^{(n-1)}$. Generally, for the i^{th} term, $a_i = a_0 \cdot r^i$, so if you know accurate values for a_0 and r, you can find any term. For example, if a_0 is exactly 180 and r is exactly 1.95, the first 4 terms (that is, $n = 4$) of the geometric sequence are 180, 351, 684.45, and 1334.6775.

Rounding to *significant digits* is a different method of rounding than the traditional, fixed-decimal/fixed-precision rounding that you might be used to. *Significant digits* are the digits of a number, in standard decimal notation, starting at the leftmost nonzero digit (the one in the largest place after all leading zeroes, even zeroes after the decimal point) and continuing to the right for exactly the next **D** digits, for some specified value of **D**. At the D^{th} digit, standard rounding rules are applied to remove all remaining digits (if after the decimal point) or to convert them to zeroes (if before the decimal point).

This method, unlike traditional rounding, does not always result in the same fixed number of digits after the decimal point, and it may result in mandatory trailing zeroes, which indicates that **D** digits (including those trailing zeroes) are significant. The decimal point itself is only included when there are significant digits after it, and a leading zero is shown before the decimal point only when the first significant digit is after the decimal point.

Here are some examples of numbers rounded to significant digits, according to the rules above, with different values for **D**:

Original Number	**Rounded to *D*=3 significant digits**	**Rounded to *D*=4 significant digits**	**Rounded to *D*=5 significant digits**
13.249999	13.2	13.25	13.250
13.25	13.3	13.25	13.250
987654321	988000000	987700000	987650000
0.01234567	0.0123	0.01235	0.012346
0.5000123	0.500	0.5000	0.50001
180	180	180.0	180.00
351	351	351.0	351.00
684.45	684	684.5	684.45
1334.6775	1330	1335	1334.7

Those last 4 rounding examples are the terms of the preceding example for a geometric sequence. If you are given a geometric sequence *only after* the terms have been rounded in this way (such as 180.0, 351.0, 684.5, 1335), you have only an approximation for many or all values, since many original values could yield the same rounding result. And if you weren't given the value of r, it will be difficult to extrapolate the sequence accurately, since rounding errors would accumulate rapidly. So you first need to find a way to reverse the effect of this kind of rounding—that is, to "unround" the numbers.

So how would you get from a **D** = 4 rounded value of 1335 back to exactly 1334.6775? It's not possible since any value from 1334.5 to 1335.4999... (inclusive), would round to 1335 using **D** = 4. But while you might not be able to find the exact original values with complete precision, you should be able to find, for some larger **D** value, the minimum and maximum valid values (the tightest possible upper and lower bounds) that would still satisfy the entire geometric sequence. If you can find the minimum and maximum valid values for a 0 and r in this way, it will enable Juliana to extrapolate the sequence with confidence.

Given the consecutive terms of a geometric sequence, each rounded to the same **D** significant digits, find the minimum and maximum valid values for a_0 (the first given term), each expressed with **D**+3 significant digits, and the minimum and maximum valid values for r, each expressed with **D** + 5 significant digits.

Input

The input will consist of two lines. The first line will contain only an integer D ($0 < D < 6$), and an integer N ($1 < N < 200$), separated by a single space. The second line will contain the consecutive terms, a_i ($10^{-8} < a_i < 10^8$), in order for the geometric sequence—exactly N numbers with D significant digits each. Each term will contain a decimal point, leading zero, and/or trailing zeroes only as required by the rules above. Each term will be separated from the next by a single space. The input numbers have been constructed such that the judges' solution requires no more than 13 significant digits in any intermediate calculation.

Output

Output a single line with four numbers in standard decimal notation, each separated by a single space, in this order: the minimum and maximum valid values for a_0 , rounded (or "unrounded" in this case) to $D + 3$ significant digits, and the minimum and maximum valid values for r, similarly "unrounded" to $D + 5$ significant digits. Follow the rules for significant digits, decimal points, leading zeroes, and trailing zeroes, exactly as given above.

Example

Input	Output
4 4 180.0 351.0 684.5 1335	179.9500 180.0500 1.94973304 1.95041335
3 6 12.9 13.0 13.2 13.3 13.4 13.5	12.850 12.949 1.0076924 1.0106650
1 3 300 20 1	250.0 350.0 0.0520988 0.0774597

Problem J. Jazz Enthusiast

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

Kai is listening to his favourite jazz playlist. He likes to turn on crossfading between songs, so during the last seconds of a song, it will slowly fade out while the next one fades in. This happens between any two consecutive songs, but the beginning of the first song and the end of the last song will be played normally.

Determine the total amount of time it takes Kai to listen to the whole playlist.

Input

The input consists of:

- Two integers n and c ($1 \leq n \leq 100$, $1 \leq c \leq 10$), giving the number of songs and the crossfade time in seconds.
- n lines of the form `m:ss` (`0:30` $\leq$ `m:ss` $\leq$ `9:59`), giving the song lengths (with one digit for the number of whole minutes and two digits for the remaining seconds).

Output

Output a string of the form `hh:mm:ss`, giving the total time it takes to listen through the whole playlist (with two digits for the number of whole hours, two digits for the number of remaining whole minutes, and two digits for the remaining seconds).

Example

Input	Output
3 5 3:59 0:52 9:40	00:14:21
1 10 6:00	00:06:00

Problem K. Colourful Chameleons

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

When Christian is not doing sophisticated research in software validation, he is breeding chameleons. His chameleons are of a special species and only occur in one of n distinct colours. Unlike their counterparts in nature, Christian's chameleons change their colour only under very specific circumstances. It took Christian quite a long time to develop his so-called *Crazy Chameleon Colouring Concept* (C^4) which works as follows. If he puts $n-1$ chameleons with distinct colours in a special breeding terrarium, gives them strawberry cheese and liver sausage to eat, and lets them stay in the dark over night, then there will be y chameleons in the terrarium the next morning. The crazy thing is that none of the y chameleons has any of the $n-1$ original colours. Instead, all of them take on the colour that was not present in the breeding terrarium initially.

Last weekend Christian let a friend borrow his chameleons for the Christopher Street Day. Of course his friend wanted the chameleons to look as colourful as possible. That's why Christian applied the C^4 method in a way such that there were at least $n-1$ chameleons available of every single colour. Therefore, Christian currently has $x_i \geq n-1$ chameleons of the ith colour. However, the next holiday coming up is St. Patrick's Day and Christian thinks that it would be super cool if all his chameleons took on the same colour for this special event. He is wondering whether such a state can be reached by exclusively applying the C^4 method. Because he is running low on strawberry cheese, he would like to know the minimal number of C^4 applications needed for all his chameleons to take on the same colour - provided that this is even possible.

Input

The input consists of:

- A line with three integers n, c, and y:
  - n ($2 \leq n \leq 10^5$), the number different colours that the Chameleons can take on.
  - c ($1 \leq c \leq n$), the colour that all chameleons should have in the end.
  - y ($n-1 \leq y \leq 10^9$), the number of chameleons in the breeding terrarium after applying the C^4 method.
- A line with n integers $x_1, \ldots, x_n$ ($n-1 \leq x_i \leq 10^9$ for all i), where x_i is the number of chameleons of the ith colour that Christian possesses initially.

Output

If it is impossible for Christian's chameleons to all take on colour c by exclusively applying the C^4 method, output `impossible`. Otherwise output two integer numbers a and b, where a is the minimal number of C^4 applications necessary for all chameleons to have colour c, and b is the total number of chameleons that Christian possesses in the end.

Example

Input	Output

`3 2 2`
`2 3 5`
	`5 10`

`3 1 3`
`2 2 3`
	`impossible`

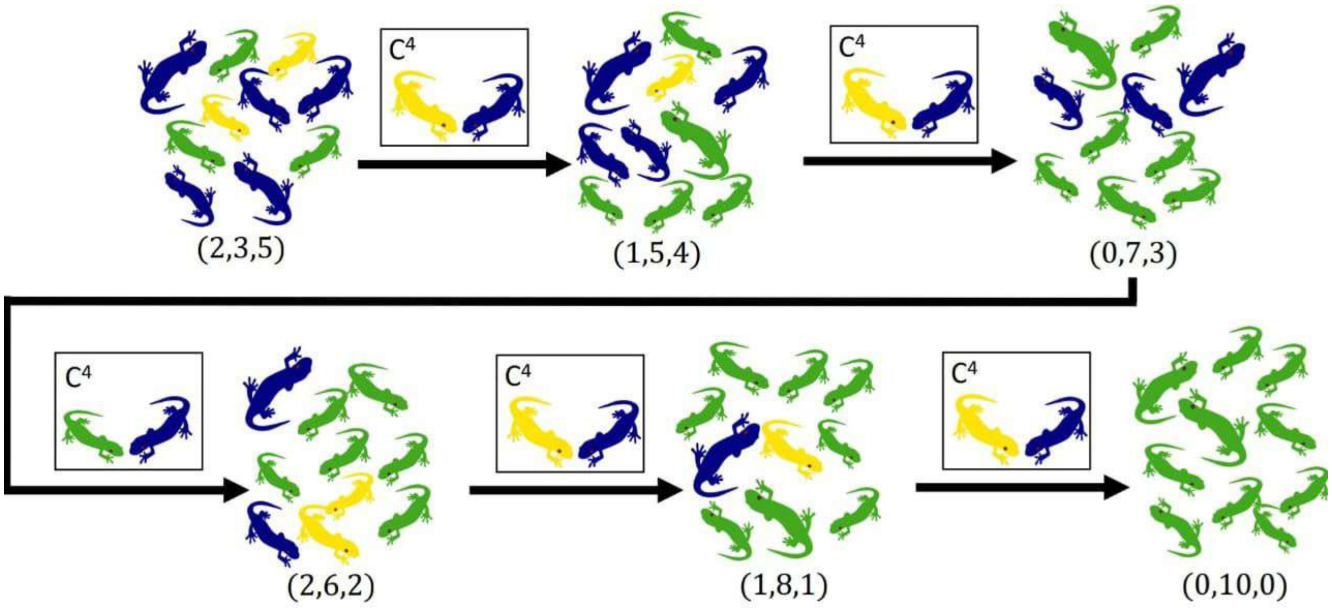


Illustration of the first sample case. Initially, Christian owns 2 yellow, 3 green, and 5 blue chameleons. After 5 applications of the C^4 method he has 10 green chameleons. The vectors (y, g, b) denote the number of yellow, green, and blue chameleons available in the respective breeding step.

Problem L. Election Meddling

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

Your university definitely needs a new building solely dedicated to computer science. The department has the money for it, but the city council does not want to approve the building permit for the new building. That's unfortunate!

Luckily, council elections are right around the corner. You have decided to participate in that election with the Interest Council Party for Computer science (ICPC). Once you have a majority of the votes in the council, you and your party colleagues can approve the new building.

The council elections are held in a number of districts to which the voters are assigned. Each district elects one council member. Each party fields one candidate per district and every voter has only one vote. The candidate with the most votes in each district will be elected as a council member.

Since the new building is very important to you and your party colleagues, you don't want to take any chances. In case of a tie in the council or in an election in a district no one knows what happens. So your objective is to win an outright majority in the council – i.e. strictly more than half of the members. Further, in each district you consider won, the ICPC candidate must receive strictly more votes than any other candidate.

You and your colleagues of the ICPC have created an incredibly sophisticated simulation of the election. Thus, you know exactly how many voters will vote for each candidate in every district. Sadly, your model does not predict a victory for the ICPC in the election. Here comes the tricky question: How many voters do you have to bribe at least in order to win the election? If you bribe a voter, he will change his previously preferred party (which you know due to your meticulous modelling) to the ICPC. People who didn't want to vote cannot be bribed.

Input

The input consists of:

- One line with two integers w and p ($2 \leq w, p \leq 1\,000$), the number of districts and the number of parties running in the election. The parties are numbered 1 to p and the ICPC is party 1.
- w lines, each with p integers $v_1, \ldots, v_p$ ($0 \leq v_i \leq 1\,000$ for each i) giving the projected results for a district. v_i denotes the number of votes that will be cast for party i.

It is guaranteed that there is at least one voter in each district, i.e. the sum of all v_i per district will always be at least one.

Output

Output the minimum number of voters that have to be bribed in order for the ICPC to win a majority of the seats in the council.

Example

Input	Output
3 3	4
0 2 2	
1 2 3	
0 2 3	
2 4	5
0 1 2 9	
1 0 0 0	
3 3	6
1 0 0	
0 1000 0	
0 9 5	