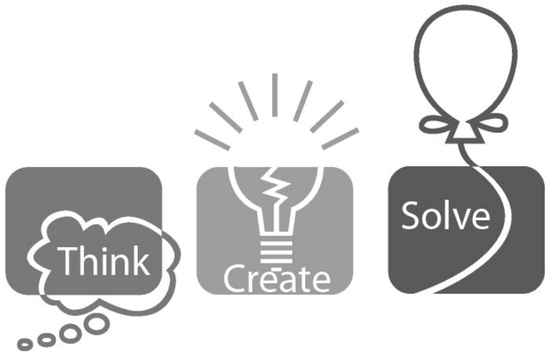
University of Central Florida



2019 (Fall) Local Programming Contest

Problems

Problem#	Difficulty Level	Filename	Problem Name
1	Easy	divide	Divide the Cash
2	Easy	freq	Sort by Frequency
3	Easy-Medium	bed	Fitting on the Bed
4	Easy-Medium	candy	Candy Land
5	Medium	gnocchi	Give-a-Gnocchi
6	Medium	moress	More or Less
7	Medium-Hard	escape	Cooperative Escape
8	Medium-Hard	view	Mountain View
9	Medium-Hard	unround	Floating-Point Unrounding
10	Hard	will	Programming Team's Will
11	Hard	match	Code Matching

Call your program file: filename.c, filename.cpp, filename.java, or filename.py

For example, if you are solving Sort by Frequency:

Call your program file: freq.c, freq.cpp, freq.java, or freq.py

UCF Local Contest — September 7, 2019

Divide the Cash

filename: `divide`
Difficulty Level: `Easy`

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.

The Problem:

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

The 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.

The Output:

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

Sample Input	Sample Output
`3 1000`	`250`
`5`	`400`
`8`	`350`
`7`	
`1 500`	`500`
`10`	

Sort by Frequency

filename: `freq`
Difficulty Level: `Easy`

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.

The Problem:

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.

The 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).

The Output:

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

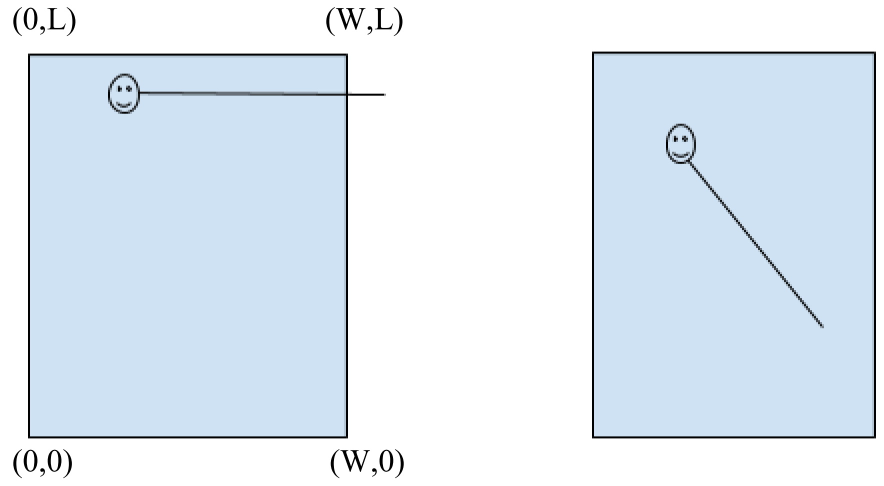
Sample Input	**Sample Output**
`dcbbdabb`	`bbbbddac`
`programming`	`ggmmrrainop`

UCF Local Contest — September 7, 2019

Fitting on the Bed

filename: `bed`
Difficulty Level: `Easy-Medium`

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!



The Problem:

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.)

The Input:

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

The Output:

On a line by itself, output "YES" (no quotes) 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.

Sample Input	**Sample Output**

`200 150 115`
`40 190 0`
	`NO`

`200 150 115`
`40 150 315`
	`YES`

Sample 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.

UCF Local Contest — September 7, 2019

Candy Land

filename: `candy`
Difficulty Level: `Easy-Medium`

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!

The Problem:

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.

The Input:

The first line of input contains two positive integers: n ($2 \le n \le 200$), the number of squares on the Candy Land board and p ($2 \le p \le 6$), 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 substring "SPECIAL").

The input line following the board definition will contain a single positive integer, c ($c \le 500$), 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.

The 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.

Sample Input	**Sample 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`

Sample 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.

UCF Local Contest — September 7, 2019

Give-a-Gnocchi

filename: `gnocchi`
Difficulty Level: `Medium`

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 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 n; on the second day, you would give the next composite number that is not divisible by a prime lower than or equal to n, etc.

The Problem:

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 n and a day k, and it will output the k^{th} (1-indexed) composite number that is not divisible by a prime less than n.

The Input:

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

The Output:

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

Sample Input **Sample Output**

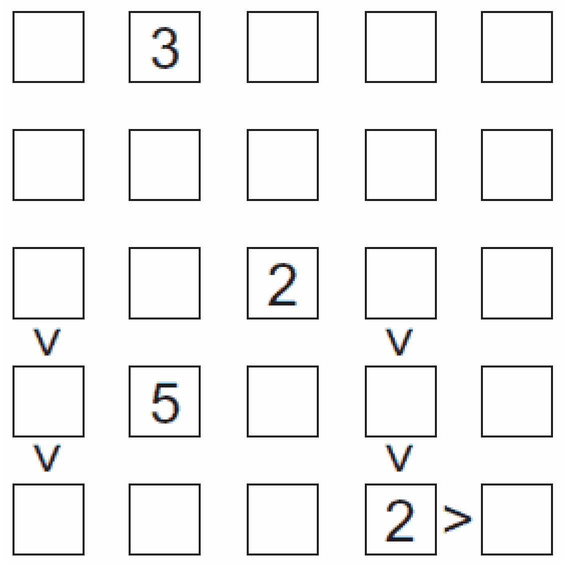
Sample Input	Sample Output
`10 3`	`169`
`1 1`	`4`
`19 7`	`943`

Sample 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 1st 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 7th value in the list is 943, which will be the value given on the 7th day.

More or Less

filename: `moress`
Difficulty Level: `Medium`

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[1]", 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 x 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.

[1] This type of puzzle already exists. It also goes by the name "Futoshiki".

The Problem:

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.

The 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.

The Output:

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

Sample Input	**Sample Output**

`5`
`-.3.-.-.-`
`.........`
`-.-.-.-.-`
`.........`
`-.-.2.-.-`
`v.....v..`
`-.5.-.-.-`
`v.....v..`
`-.-.-.2>-`
	`23415`
	`12354`
	`51243`
	`45132`
	`34521`

`4`
`-.-.-.-`
`..^....`
`-.-.->-`
`.......`
`-<-.-.-`
`v.^....`
`-.-.-.2`
	`4123`
	`3241`
	`2314`
	`1432`

UCF Local Contest — September 7, 2019

Cooperative Escape

filename: `escape`
Difficulty Level: `Medium-Hard`

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 Problem:

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.

The Input:

The first input line contains two integers, r and c ($2 \le r, c \le 30$), 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).

The 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.

Sample Input **Sample Output**

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

```
9
```

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

```
-1
```

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

```
14
```

Sample 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.

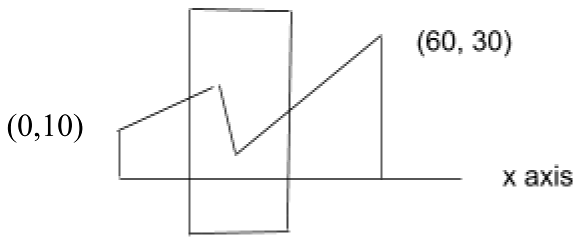
UCF Local Contest — September 7, 2019

Mountain View

filename: `view`
Difficulty Level: `Medium-Hard`

You have spent the summer in Mountain View and want to share with your friends the beautiful view. The mountains are far away enough that we can model them as lined up on a straight line on the positive x-axis. When you take a picture of the mountains, your camera captures a fixed width, W, of the mountain view, capturing the mountain outline in the range [x, x+W].

If we assign the elevation of a point to be its y-value, then we can describe the mountain outline as a series of ordered pairs, from left to right, where the mountain line between ordered pairs is the straight line between the two points. Consider the mountain outline defined by the points (0, 10), (20, 20), (30, 5), and (60, 30):



PIcture of width W = 20

If your camera width was W = 20, then one possible picture you could take is the one shown above in between the range of x = 15 and x = 35. Since you want to show your friends a mountainous view, you would like to show them a picture where the average height of the mountain is maximal of all possible pictures you could take with a fixed width of W. For the example above, the best choice would be the picture from x = 40 to x = 60. For this picture, the average elevation of the mountain line is y = 65/3.

The Problem:

Given the description of a mountain line and a fixed width W for your picture, determine, of all possible pictures you could take, the maximum average elevation.

The Input:

The first line of input contains two positive integers: n ($2 \le n \le 10^5$), the number of points describing the mountain line, and W ($1 \le W \le 10^9$), the width of the picture your camera takes. The ith ($1 \le i \le n$) line of the following input lines contains two non-negative integers, x_i ($x_i \le 10^9$) and y_i ($y_i \le 10^4$), the x and y coordinates, respectively, of the ith point describing the skyline. It is guaranteed for all $i < n$ that $x_i < x_{i+1}$. Assume that outside of the mountain landscape given in the input, the elevation is always 0.

The Output:

On a line by itself, output the maximum average height of all possible pictures you could take. Print 9 digits after the decimal point. Answers will be judged correct if they are within an absolute or relative tolerance of 10^{-6}.

Sample Input	**Sample Output**

`4 20`
`0 10`
`20 20`
`30 5`
`60 30`
	`21.666666667`

`3 1`
`10 50`
`90 50`
`1000 49`
	`50.000000000`

Floating-Point Unrounding

filename: `unround`
Difficulty Level: `Medium-Hard`

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, … $a_{n\text{-}1}$, where the terms (i.e., the numbers in the sequence) are related by powers of a constant factor r, specifically, $a_1 = a_0r$, $a_2 = a_0r^2$, $a_3 = a_0r^3$, …, and $a_{n\text{-}1} = a_0r^{(n\text{-}1)}$. Generally, for the ith term, $a_i = a_0r^i$, 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 $\boldsymbol{D} = 4$ 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 $\boldsymbol{D} = 4$. 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 $\boldsymbol{D}$ 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.

The Problem:

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

The 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.

The 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.

Sample Input **Sample 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

Programming Team's Will

filename: `will`
Difficulty Level: `Hard`

The UCF Programming Team has a not-so-well-known tradition of creating "Last Wills" for the event that one of the members leaves the team during the year unannounced. If that happens, something must be done with their lollipop collection in the Programming Team Lab! The Programming Team lets its members specify to which other UCF students they want to give their lollipops. Each member writes up a list of UCF students and, for each student, what fraction of the lollipop collection should be given to them. Note that all team members are UCF students as well so a team member could have another member in their will.

Normally this is a perfect solution; someone drops out of UCF to switch to some lesser college, and their lollipops are spread among their specified recipients. However, there has been a horrible tragedy. Some of the team was enrolled in Dr. Meade's CS-I class, and just took the first exam. Everyone who was enrolled has failed out of the class, and is kicked off the team by Dr. Meade! But what should happen to their lollipop collections now? The issue is that some team members had each other in their wills, so we cannot simply resolve them one at a time.

To handle this, the team agreed upon a fair strategy to distribute the lollipops. The wills would all be repeatedly applied (including giving lollipops to departing team members) until the total lollipops of departing members converged to zero. In the event that infinite applications of everyone's wills would still leave some lollipops with departing members, those lollipops would be thrown away. However, this is potentially a very slow and infinite process, so they need your help to figure out where the lollipops end up. Note that a fractional part of a lollipop can be given, if needed, by crushing up the lollipop.

The Problem:

Given a list of departing team members' wills and their lollipop counts, output how many lollipops each student at UCF will end up with.

The Input:

The first input line contains two space separated positive integers: N ($2 \le N \le 500$) representing the number of programming team members in Dr. Meade's class, and M ($N < M \le 50{,}000$) representing the total number of UCF students. The programming team members in Dr. Meade's class have ID's 1 through N, and the other students have ID's N+1 through M. Then follow N wills, each described as follows:

Each will starts with a line containing two integers: L ($1 \le L \le 1000$) representing the number of lollipops, and K ($1 \le K < M$) the number of entries in this will. The following K lines each contain an integer X specifying the id of the person in the will, and a floating-point number P ($P \ge 10^{-6}$)

specifying what portion (fraction) of the lollipop collection goes to person *X* in this will. Assume that:

- A single person's will contains only unique people, i.e., there won't be two entries for the same person in a will.
- The will for a person will not contain themselves.
- The fractions in a team member's will add up to 1.
- The total number of entries in all wills will not exceed 1,000,000.
- The input values for *P* will be given with no more than 6 digits after the decimal point.

The Output:

Output *M* lines, containing the number of lollipops that each student ends up with (in order by ID). Note that the first *N* lines should be 0, since all lollipops from those students will either be given away or thrown away. Your answer will be judged to a precision of 10^{-5}.

Sample Input	Sample Output

2 5
100 2
2 0.9
3 0.1
100 2
1 0.2
4 0.8
	0.0
	0.0
	14.63414634
	185.36585366
	0.0

UCF Local Contest — September 7, 2019

Code Matching

filename: `match`
Difficulty Level: `Hard`

James is a spy, working in a country where communication back to his headquarters is not easy. He receives instructions from his superiors over a radio signal, through the use of number stations. A number station is a radio station where a list of numbers are occasionally read out, that have some encrypted meaning. James has a codebook with a mapping of number sequences (e.g., “732736”) to instructions (e.g., “abort the mission”). Every sequence of numbers listed in his codebook maps to a unique instruction. However, James is a lazy spy; he only checks in on the radio occasionally. His superiors know this, and so they assume that he will start listening to their sequence at a random digit, each equally likely (as early as the first, as late as the last). Their instructions are very important though, so it’s essential that they figure out exactly how long it will take James to decode the message. As soon as James can be positive that there is only one message that matches what he is hearing (factoring in that he may have missed some digits before he tuned in), he will stop listening and take action.

Each digit takes one second to read, and there will be silence one second after the last digit (telling James the message is complete). For each message that headquarters might send, you have to determine how long it will take James to understand the message on average. Note that it is possible that sometimes even after listening until the end of the message, James cannot be sure which message he heard. In that case, print “Impossible”.

The Problem:

Given the codebook, and the code to be sent, find the expected amount of time that James will spend listening to his radio.

The Input:

You will be given an integer, N ($\leq$ 100,000), specifying the number of sequences in James’s codebook. The next N lines will each contain one sequence, with total length not exceeding 100,000 (i.e., the length of all sequences together will not exceed 100,000).

The Output:

Output N lines. The ith line should contain the expected number of seconds James will spend listening before he can be certain of the message he is hearing if headquarters sends the ith message in his codebook. Your answer will be judged to a relative error of 10^{-5}. If it’s possible that James might not ever be able to determine the message, print out “Impossible” for that line.

Sample Input

```
4
17383
126
385
485
```

Sample Output

```
2
1.333333333
Impossible
Impossible
```

Sample Explanation: For the first message (17383), there are 5 possible spots where James could tune in. If he tunes in at the first digit, it could still be the second sequence (126) since both contain a 1. Once he hears the following 7, it could only be the first sequence. Therefore, tuning in at this digit takes 2 seconds. If he tunes in on the second digit, it takes 1 second. The third digit takes 3 seconds, the fourth digit takes 2 seconds, and the fifth digit takes 2 seconds (he hears the 3, then hears silence). This gives an average of (2+1+3+2+2)/5 = 2.
For the third sequence, if he tunes in on the 8 then he will hear "85_", and therefore cannot tell it apart from the fourth sequence. Thus the answer here is "Impossible".