Problem A. All in the Family

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

The scientists working at the Family Genetics Institute are tracing the spread of a hereditary disease through family trees. They have started by listing the family members that have the disease, and the parent who passed the disease on to each child (we will assume that each child gets the disease from only one parent). However, the scientists are confused about the names of different family relationships. Parents, grandparents, and siblings they have a handle on. But a relationship like "third cousin, twice removed" has been hard for them to wrap their heads around. After much discussion they came up with some definitions that cleared things up.

Suppose we have two people conveniently named A and B and their closest common ancestor is named C (what are the odds!). We say that A is m *generations removed from* C if there are m direct descendants from C ending with A. Thus if A is the daughter of C she is 1 generation removed; if she is the granddaughter of C she is 2 generations removed, and so on. Any person is 0 generations removed from themselves.

Now let A be m generations removed from C and B be n generations removed from C where $m \leq n$. We can determine the relationship between A and B using the following rules:

1. if $m = 0$ then B is the child of A if $n = 1$ or the great^{n-2} grandchild of A if $n > 1$.
2. if $0 < m = n$ then A and B are siblings if $n = 1$ or $(n-1)$-th cousins if $n > 1$.
3. if $0 < m < n$ then A and B are $(m-1)$-th cousins $(n-m)$ times removed.

Notice that if $m = 1$ and $n = 2$ we get the interestingly named "0th cousins, 1 time removed" for the relationships we typically describe as "aunt/uncle" or "niece/nephew".

The figure below shows some examples for two new people named (what else) D and E.

# of generations removed from common ancestor		
D	E	Relationship
0	1	E is the child of D
4	0	D is the great great grandchild of E
3	3	D and E are 2nd cousins
9	8	D and E are 7th cousins, 1 time removed
1	4	D and E are 0th cousins, 3 times removed

Some example relationships

The scientists have given you the description of a family tree and pairs of people in the tree and have asked you to determine the relationships between members of each pair.

Input

Input begins with a line containing two positive integers t p ($t \leq 100, p \leq 1\,000$) specifying the number of tree descriptions (described below) and the number of query pairs. Following these are t lines, each with one tree description. Each tree description will be of the form s_0 d s_1 $s_2 \ldots s_d$ indicating that person s_0 has d children named s_1 through s_d. All names are unique and contain only alphabetic characters. Tree descriptions may be given in any order (i.e., the root of the entire tree may not necessarily be in the very

first tree description). No name will appear more than once as s_0 in the tree descriptions. All the tree descriptions will combine to form exactly one tree, and the tree will have at least 2 nodes and at most 100 nodes.

Following this are p lines of the form s_i s_j where $s_i \neq s_j$ and both names are guaranteed to be in the tree.

Output

Output the relationship for each pair of people, one per line, using the formats shown in Figure 1. Always output s_i's name first for each pair except when s_j is the direct descendant of s_i (as in the first example in Figure 1). For the n-th ordinal number output n`th` except for $n = 1, 2, 3, 21, 22, 23, 31, 32, 33, \ldots$ in which case you should output `1st`, `2nd`, `3rd`, `21st`, `22nd`, `23rd`, `31st`, `32nd`, `33rd`, etc. Also be sure to use `times` for all times removed except one, where you should use the word `time`.

Example

Input	Output
4 5 Horatio 1 Irene Chris 2 Gina Horatio Alice 3 Dan Emily Frank Bob 2 Alice Chris Irene Bob Dan Frank Chris Emily Alice Chris Dan Irene	Irene is the great grandchild of Bob Dan and Frank are siblings Chris and Emily are 0th cousins, 1 time removed Alice and Chris are siblings Dan and Irene are 1st cousins, 1 time removed
4 6 A 4 B C D E H 3 I J K C 2 F G D 1 H G C H A F G F H F K B K	G is the child of C H is the grandchild of A F and G are siblings F and H are 1st cousins F and K are 1st cousins, 1 time removed B and K are 0th cousins, 2 times removed

Problem B. Kinky Word Search

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

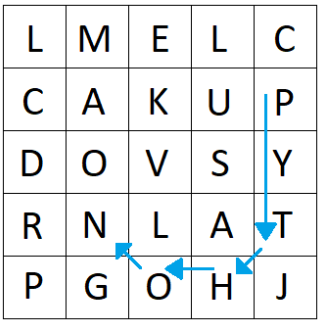
You're probably familiar with regular word searches, where you're presented with a grid of letters and a word to find. The word can be in a straight line horizontally, vertically, or diagonally (and perhaps backwards in any of those directions). For example, here is a grid of letters:

L	M	E	L	C
C	A	K	U	P
D	O	V	S	Y
R	N	L	A	T
P	G	O	H	J

A word search grid

The word "JAVA" can be found going from the bottom right corner diagonally upwards.

In a *kinky word search* the path that spells out the word can have one or more "kinks" – places where the path changes direction. For example, in the given grid you can spell the word "PYTHON" with 3 kinks (one each at the T, H and O):



A kinky spelling of "PYTHON"

Adding kinks allows letters to be reused – the word "CPLUSPLUS" can be found in the upper right corner of the grid (with 5 kinks). However you cannot stay on a letter twice in a row, so you cannot spell the word "HASKELL" in this grid (though you can find at least 11 more common programming languages). Your task is to see if the spelling of a word with a certain number of kinks is possible or not.

Input

Input begins with a line containing two positive integers r and c ($r, c \leq 10$), the number of rows and columns in the grid. After this are r rows of c uppercase characters. Letters are separated by a space. After the grid are two lines: The first line is an integer k, the number of kinks. The second line contains an uppercase word to look for, with maximum length 100.

Output

Output either the word `Yes` if it is possible to spell the given word with exactly k kinks on the grid

provided, or `No` if it is not.

Example

Input	Output
5 5 L M E L C C A K U P D O V S Y R N L A T P G O H J 0 JAVA	Yes
5 5 L M E L C C A K U P D O V S Y R N L A T P G O H J 3 PYTHON	Yes
5 5 L M E L C C A K U P D O V S Y R N L A T P G O H J 4 PYTHON	No

Problem C. Math Trade

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

Suppose a group of people have objects they want to trade, and objects they want to get in return:

Name	Has	Wants
Sally	Clock	Doll
Steve	Doll	Painting
Carlos	Painting	Clock
Maria	Candlestick	Vase

Notice how none of the people listed can pair off and trade with each other. However, if Sally, Steve, and Carlos all got together, Sally could trade her clock to Steve for the doll she wants, and Steve can then trade that clock to Carlos for the painting Steve wants.

This creation of a chain of individual trades that gets a large number of people the objects they want is called a *math trade*. Ideally everyone is involved in the math trade but that is not always possible (sorry Maria). The object therefore is to create a single chain of the longest length. Determining the longest math trade gets complicated if the participants have multiple items that they want to trade or obtain. Luckily for you, we are only worried about the situation where each person has and wants exactly one item, and no item is owned by or desired by more than one person.

Input

Input begins with a line containing a positive integer $n\,(n \leq 100)$, the number of people interested in trading. After this are n lines, each with three strings separated by spaces. The first string will be the name of the trader. The second string will be the object the trader has. The third string will be the object the trader wants. All trader names will be unique, and no object will be wanted by more than one person and owned by more than one person.

Output

Output the length of the longest math trade. If no trading is possible, output the phrase "No trades possible"

Example

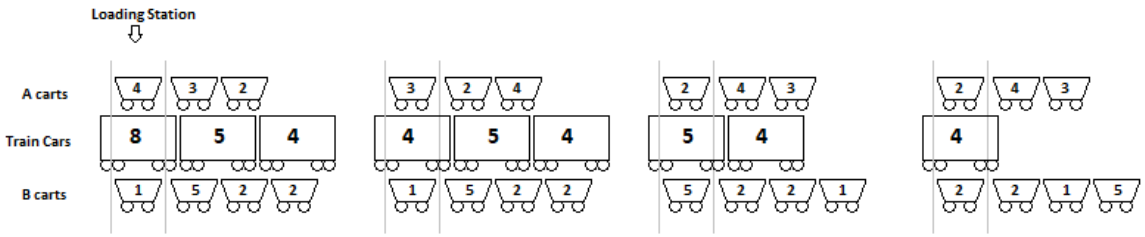
Input	Output
4 Sally Clock Doll Steve Doll Painting Carlos Painting Clock Maria Candlestick Vase	3
4 Abby Bottlecap Card Bob Card Spoon Chris Spoon Chair Dan Pencil Pen	No trades possible

Problem D. Oreperations Research

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

You run a massive mining facility that extracts ore from a mine and loads it onto long trains for shipment to factories around the nation. A train consists of n cars, where car i has capacity c_i, indicating how many tons of ore it can carry. Ore is dropped into the train cars at an overhead loading station from two separate queues of mine carts that run on either side the train. As with the train cars, mine carts hold varying loads of ore. Queue A has r carts, and cart A_i carries a load of a_i. Similarly, queue B has s carts, and cart B_i carries a load of b_i. Initially train car 1 is at the loading station, and carts A_1 and B_1 are available to dump ore into it. The train car currently in the station can be given the load from the front cart in the A queue, the front cart in the B queue, or from both. If a cart doesn't dump its ore, it remains at the loading station; if it does dump its ore, it cycles back into the mine, loads up on ore, and rejoins the end of its queue. Meanwhile, the next cart in the queue moves into place and is available to dump ore. Carts may not drop partial loads of ore and may not leave the loading station until they've emptied. Similarly train cars may not be over-filled and may not leave the loading station until they are filled to capacity. As soon as a train car is filled to capacity it leaves the loading station and the next train car pulls in. Your task is to determine whether given sequences of mine carts can be used to fill a given sequence of train cars to their capacity.

The figure shows an example of the process. Here queue A has three mine carts carrying loads 4, 3 and 2, queue B has four mine carts carrying loads 1, 5, 2 and 2, and the train has three cars with capacities 8, 5 and 4. The starting setup is shown in the leftmost image. After (say) the first car in queue A dumps its load into the first train car it goes back to the mine and (eventually) returns to the end of the line in queue A. This situation is shown in the second image, where the first train car still has capacity 4 to be filled. This can be accomplished by dumping ore from the front car of both queues A and B. Once filled, the first train car moves out of the loading station leaving an alignment of cars and carts shown in the third image. Here, the only way to fill the train car is for the front car of queue B to dump its load. This leads to the final image. Here the last train car can be filled either by the front cars of both queues or the first two cars of queue B. Note that if the third train car had capacity 3 it could not be filled to full capacity.



Sample Input 1

Input

Input begins with a line containing three positive integers r s n $(r, s \leq 50, n \leq 100)$ indicating the number of carts in queues A and B and the number of train cars, respectively. This is followed by three lines containing the values $a_1, a_2, \ldots, a_r$, $b_1, b_2, \ldots, b_s$, and $c_1, c_2, \ldots, c_n$, the capacities of the A carts, the B carts and the train cars, respectively. The maximum capacity of any cart is 200 and the maximum capacity of any train car is 2 000 000.

Output

Output `Yes` or `No` indicating whether all of the train cars can be filled to capacity.

Example

Input	Output
3 4 3 4 3 2 1 5 2 2 8 5 4	Yes
3 4 3 4 3 2 1 5 2 2 8 5 3	No

Problem E. Over the Hill, Part 1

Source file name:	Overthehill01.c, Overthehill01.cpp, Overthehill01.java, Overthehill01.py
Input:	`Standard`
Output:	`Standard`

Hill encryption (devised by mathematician Lester S. Hill in 1929) is a technique that makes use of matrices and modular arithmetic. It is ideally used with an alphabet that has a prime number of characters, so we'll use the 37 character alphabet `A, B, ..., Z, 0, 1, ..., 9,` and the space character. The steps involved are the following:

1. Replace each character in the initial text (the *plaintext*) with the substitution `A`$\rightarrow 0$, `B`$\rightarrow 1$, ..., `(space)`$\rightarrow 36$. If the plaintext is `ATTACK AT DAWN` this becomes

$$0\ 19\ 19\ 0\ 2\ 10\ 36\ 0\ 19\ 36\ 3\ 0\ 22\ 13$$

2. Group these number into three-component vectors, padding with spaces at the end if necessary. After this step we have

$$\begin{pmatrix} 0 \\ 19 \\ 19 \end{pmatrix} \begin{pmatrix} 0 \\ 2 \\ 10 \end{pmatrix} \begin{pmatrix} 36 \\ 0 \\ 19 \end{pmatrix} \begin{pmatrix} 36 \\ 3 \\ 0 \end{pmatrix} \begin{pmatrix} 22 \\ 13 \\ 36 \end{pmatrix}$$

3. Multiply each of these vectors by a predetermined 3×3 encryption matrix using modulo 37 arithmetic. If the encryption matrix is

$$\begin{pmatrix} 30 & 1 & 9 \\ 4 & 23 & 7 \\ 5 & 9 & 13 \end{pmatrix}$$

then the first vector is transformed as follows:

$$\begin{pmatrix} 30 & 1 & 9 \\ 4 & 23 & 7 \\ 5 & 9 & 13 \end{pmatrix} \begin{pmatrix} 0 \\ 19 \\ 19 \end{pmatrix} = \begin{pmatrix} (30 \times 0 + 1 \times 19 + 9 \times 19) & \mod 37 \\ (4 \times 0 + 23 \times 19 + 7 \times 19) & \mod 37 \\ (5 \times 0 + 9 \times 19 + 13 \times 19) & \mod 37 \end{pmatrix}$$
$$= \begin{pmatrix} 5 \\ 15 \\ 11 \end{pmatrix}$$

4. After multiplying all the vectors by the encryption matrix, convert the resulting values back to the 37-character alphabet and concatenate the results to obtain the encrypted *ciphertext*. In our example the ciphertext is `FPLSFA4SUK2W9K3`.

This method can be generalized to work with any $n \times n$ encryption matrix in which case the initial plaintext is broken up into vectors of length n. For this problem you will be given an encryption matrix and a plaintext and must compute the corresponding ciphertext.

Input

Input begins with a line containing a positive integer $n \leq 10$ indicating the size of the matrix and the vectors to use in the encryption. After this are n lines each containing n non-negative integers specifying the encryption matrix. After this is a single line containing the plaintext consisting only of characters in the 37-character alphabet specified above.

Output

Output the corresponding ciphertext on a single line.

Example

Input	Output
3 30 1 9 4 23 7 5 9 13 ATTACK AT DAWN	FPLSFA4SUK2W9K3
6 26 11 23 14 13 16 6 7 32 4 29 29 26 19 30 10 30 11 6 28 23 5 24 23 6 24 1 27 24 20 13 9 32 18 20 18 MY HOVERCRAFT IS FULL OF EELS	W4QVBOONJG5 Y76H5A6XHR11BV670Z

Problem F. Over the Hill, Part 2

Source file name:	Overthehill02.c, Overthehill02.cpp, Overthehill02.java, Overthehill02.py
Input:	`Standard`
Output:	`Standard`

Bob Roberts is part of a crack espionage team working for the CIA (Chocolate Institute of Alabama) and he is working on decrypting the encoded messages of their arch rivals at the NSA (Nougat Society of Arkansas). Fortunately, the NSA's espionage staff is not nearly as crack as Bob's as they are using the Hill encryption scheme (described in the previous problem) which is susceptible to a known-plaintext attack. Bob has intercepted a plaintext/ciphertext pair and has knowledge of the size of the encryption matrix used by his not-so-sweet enemies. Given these Bob knows that there is a way to determine the encryption matrix, but no one on his staff is exactly sure how (hmmm ... not quite as crack as they thought). Bob's come to you to solve this problem for them. One complication is that there might not be enough data to uniquely determine the NSA's encryption matrix, and the data they intercepted might have been corrupted leading to no solution to the problem.

Input

Input begins with a line containing a positive integer $n \leq 10$ indicating the size of the matrix and the vectors to use in the encryption. After this are two lines: the first of these contains the plaintext and the second the ciphertext. Both of these lines will consist only of characters in the set `A`, ..., `Z`,`0`, ..., `9` and the space character. The lengths of both strings are identical and are multiples of n. Both of these strings may include trailing blanks.

Output

Output one of three possible answers. If the input does not admit any possible encryption matrix output `No solution`. Otherwise if the input does not uniquely determine the encryption matrix output `Too many solutions`. Otherwise output the encryption matrix, one row per line with a single space between values on a line.

Example

Input	Output
3 ATTACK AT DAWN FPLSFA4SUK2W9K3	30 1 9 4 23 7 5 9 13
3 ATTACK FPLSFA	Too many solutions
3 ATTACK AT DAWN EPLSFA4SUK2W9K3	No solution

Problem G. A Rank Problem

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

Coach is fed up with sports rankings – he thinks those who make up these bogus orderings are just nuts. In Coach's opinion changes in rankings should be evidence-based only. For example, suppose the 4th place team plays the 1st place team and loses. Why should the rankings be altered? The "worse" team lost to the "better" team, so nothing should change in the rankings. Put another way, there's no evidence that the ordering should change so why change it? The only time you change something is if, say, the 4th place team beats the 1st place team. NOW you have evidence that the rankings should change! Specifically, the 1st place team should be put directly below the 4th place team (we now have evidence that backs this up) and the teams in 2nd through 4th place should each move up one. The result is that the former 1st place team is now in 4th, one position below the team that beat it, the former 4th place team now in 3rd. Note that the relative positions of the teams now in 1st to 3rd place do not change – there was no evidence that they should.

To generalize this process, assume the team in position n beats the team in position m. If $n < m$ then there should be no change in the rankings; if $n > m$ then all teams in positions $m+1, m+2, \ldots, n$ should move up one position and the former team in position m should be moved to position n.

For example, assume there are 5 teams initially ranked in the order T1 (best), T2, T3, T4, T5 (worst). Suppose T4 beats T1. Then as described above the new rankings should become T2, T3, T4, T1, T5. Now in the next game played let's say T3 beats T1. After this the rankings should not change – the better ranked team beat the worse ranked team. If in the next game T5 beats T3 the new rankings would be T2, T4, T1, T5, T3, and so on.

Coach was all set to write a program to implement this scheme but then he heard about ties in the English Premier League. The last we saw him he was standing motionless, staring out of his window. We guess it's up to you to write the program.

Input

Input begins with a line containing two positive integers n m $(n, m \leq 100)$ indicating the number of teams and the number of games played. Team names are T1, T2, ..., Tn and initially each team Ti is in position i in the rankings (i.e., team T1 is in 1st place and team Tn is in last place). Following the first line are m lines detailing a set of games in chronological order. Each of these lines has the form Ti Tj $(1 \leq i, j \leq n, i \neq j)$ indicating that team Ti beat team Tj.

Output

Output a single line listing the final ranking of the teams. Separate team names with single spaces.

Example

Input	Output
5 3 T4 T1 T3 T1 T5 T3	T2 T4 T1 T5 T3
8 4 T4 T1 T1 T2 T2 T3 T3 T4	T1 T2 T3 T4 T5 T6 T7 T8

Problem H. Restroom Monitor

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

Irma P. Freely (yes, we've hit a new low) is in charge of the bank of restrooms at the Rest Pit truck stop. Every so often a tour bus stops by and a load of passengers gets off to use the restroom. Irma has a set of n single-stall restrooms she can allocate to people. Everyone takes the same amount of time to use the facilities but may have different "deadlines" for when they must be finished. Being good at her job, Irma can look at the people in line and estimate with complete accuracy what their deadlines are.

Unfortunately, owing to shortages caused by the COVID pandemic, there is only one roll of toilet paper. Not everyone needs toilet paper, but only one person can be in a stall with it at a time. Irma needs to figure out whether she can schedule everyone so that they can all make use of her facilities before their deadlines. Sounds like a lot of paperwork ... can you help her?

Input

Input begins with a line containing two integers s and n, where $1 \leq s \leq 50\,000$ is the number of stalls and $1 \leq n \leq 100\,000$ is the number of people who need to use a restroom. Following this are n lines, one for each person. Each line contains an integer d, $1 \leq d \leq 10^9$ (the deadline) and a character t. If t is `y`, then this person needs the toilet paper; if t is `n`, they don't. Assume each user requires one unit of time and that deadlines are specified in terms of the same unit.

Output

If it is possible to allocate everyone to stalls to meet their deadlines, display `Yes`. Otherwise, display `No`.

Example

Input	Output
3 7 2 y 2 n 5 y 1 n 5 n 2 y 1 n	Yes
2 7 2 y 2 n 5 y 1 n 5 n 2 y 1 n	No

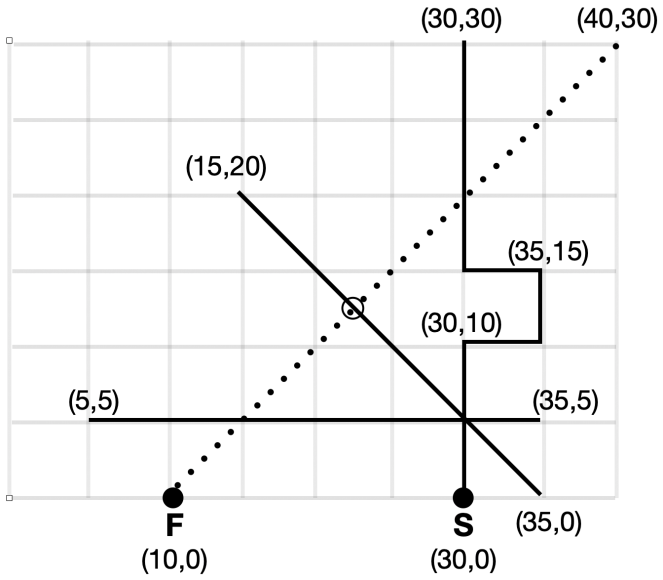
Problem I. Scholar's Lawn

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

Famously at Cambridge, and often copied at other schools, is the tradition of the "Scholar's Lawn" – an area of grass where Fellows of the school, or other distinguished entities, can walk, but regular students cannot.

So, if a student spies a Fellow walking across campus, and wishes to ambush . . . , er, meet up with them, the student is restricted to walking along a set of narrow paved walkways laid out in various places within the grassy areas, hoping to reach the Fellow's path at the same time or before the Fellow arrives. At the end of the Fellow's path is the Sacred Grove of Academe, off-limits to students, so if the Fellow reaches it before the student, the student is out of luck.

For instance, the Figure shows an area of lawn together with the fixed set of paved walkways (solid lines) and the path taken by a Fellow of the university (dotted line); F and S denote the initial positions of the Fellow and student, respectively. If both travel at the same speed (say, one meter per second), then after 17.67767 seconds the Fellow will find the student waiting to have a chat at location $(22.5, 12.5)$ (marked by the small open circle "○").



Sample Input 1

Input

Input begins with an integer n, $1 \leq n \leq 500$, the number of straight-line walkways. There will then follow n lines, each with 4 integers, denoting the (x, y) coordinates of the endpoints of each walkway. After that is a line containing three real values x_s, y_s, v_s, where (x_s, y_s) is the position of the student and v_s, $0 < v_s \leq 1\,000$, is the student's walking speed. The point (x_s, y_s) is guaranteed to lie on one of the n paved walkways. The final line contains 5 numbers. The first 4 numbers are real numbers x_{1f}, y_{1f}, x_{2f}, y_{2f}, $-10\,000 \leq x_{1f}, y_{1f}, x_{2f}, y_{2f} \leq 10\,000$, giving the starting position (x_{1f}, y_{1f}) of the Fellow and the ending position (x_{2f}, y_{2f}) of the Fellow (the last point where the student can reach the Fellow). The final number is a real value v_f, $0 < v_f \leq 1\,000$, giving the Fellow's walking speed. All real-valued inputs will have at most four digits after the decimal point.

The Fellow always walks in a straight line. The student can walk only along walkways, which are assumed

to have zero width. If a walkway intersects with another walkway or the Fellow's path, it will do so at a single point. Collinear walkways never intersect one another; similarly, if the Fellow's path and a walkway are collinear, they will not intersect.

Output

Output the earliest time t when the student's position and the Fellow's position can coincide at an intersection of a walkway and the Fellow's path. If this is impossible, output the word "`Impossible`". Numeric answers should be accurate to within an absolute error of 10^{-6}.

Example

Input	Output
7 5 5 35 5 35 0 15 20 30 0 30 10 30 10 35 10 35 10 35 15 35 15 30 15 30 15 30 30 30.0 0.0 1.0 10 0 40 30 1.0	17.67766953
7 5 5 35 5 35 0 15 20 30 0 30 10 30 10 35 10 35 10 35 15 35 15 30 15 30 15 30 30 30.0 0.0 1.0 10 0 40 30 1.2	Impossible

Problem J. Simply Sudoku

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

Sudoku puzzles come in all different shapes and difficulty levels. Traditionally a Sudoku puzzle is a 9×9 grid. Initially, some of the cells are filled in with numbers and some are empty. The goal is to fill in each cell with a number in the range $1 - 9$ subject to the following restrictions:

- Each digit $1 - 9$ must appear once in each row
- Each digit $1 - 9$ must appear once in each column
- Each digit $1 - 9$ must appear once in each 3×3 sub-grid

The difficulty of a Sudoku puzzle can vary widely. The easiest puzzles can be solved with the following two simple techniques:

Single Value Rule: search for squares which only have one possible value that can be put there.

Unique Location Rule: within any row, column or sub-grid search for a value that can only be placed in one of the nine locations.

Consider the partially solved Sudoku puzzle shown in Figure 6. The Single Value Rule applies to grid square A7 where 8 is the only value that can be placed there. The Unique Location Rule can be used to put a 5 in square B3 as it is the only location in row 3 where a 5 can be placed.

	A	B	C	D	E	F	G	H	I
9	2	6		5	1		3		
8	3				6				2
7		1	5		7	3	9		4
6			9				5		
5			2	6		1	4		
4			6				7		
3	6		1	9	4		2	3	
2	9				2				5
1			8		3	5		4	9

Sample Input 1

The easiest Sudoku puzzles can be solved with only these two rules; harder puzzles use techniques like swordfish, x-wings and BUGs.

For this problem you will be given a Sudoku puzzle and must determine if it is an easy puzzle, i.e., whether it can be solved by just using the Single Value and Unique Location rules.

Input

Input consists of a single Sudoku puzzle given over nine lines. Each line contains 9 values in the range 0 to 9, where a 0 indicates a blank in the puzzle.

Output

Output the word `Easy` followed by the solved Sudoku puzzle if it is an easy puzzle. The puzzle should be printed on nine lines with a single space separating values on a line. If the puzzle is not easy output `Not easy` followed by as much of the Sudoku puzzle as can be solved using the two rules described above. Use the same format for the partial solution as for the complete solution using a '.' instead of a digit for a unfilled square.

Example

Input	Output
2 6 0 5 1 0 3 0 0 3 0 0 0 6 0 0 0 2 0 1 5 0 7 3 9 0 4 0 0 9 0 0 0 5 0 0 0 0 2 6 0 1 4 0 0 0 0 6 0 0 0 7 0 0 6 0 1 9 4 0 2 3 0 9 0 0 0 2 0 0 0 5 0 0 8 0 3 5 0 4 9	Easy 2 6 4 5 1 9 3 7 8 3 9 7 8 6 4 1 5 2 8 1 5 2 7 3 9 6 4 1 3 9 4 8 7 5 2 6 5 7 2 6 9 1 4 8 3 4 8 6 3 5 2 7 9 1 6 5 1 9 4 8 2 3 7 9 4 3 7 2 6 8 1 5 7 2 8 1 3 5 6 4 9
0 0 0 0 0 0 7 0 1 0 0 0 0 0 1 2 3 5 0 0 1 8 0 0 0 0 6 0 0 0 0 2 5 0 9 3 9 0 0 0 0 0 0 0 2 3 1 0 6 7 0 0 0 0 2 0 0 0 0 3 8 0 0 1 3 8 9 0 0 0 0 0 4 0 6 0 0 0 0 0 0	Not easy . . 3 . . . 7 8 1 . . . . . 1 2 3 5 . . 1 8 3 . 9 4 6 . . . . 2 5 . 9 3 9 . . 3 . . . 7 2 3 1 2 6 7 9 4 5 8 2 . . . . 3 8 . . 1 3 8 9 . . 5 . . 4 . 6 . . . 3 . .

Problem K. Weighty Tomes

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

The Computer Science Library on campus needs to be temporarily closed for renovations and you have been put in charge of the storage of all the library's books. After getting over the shock of being selected as well as the shock that the campus actually had a CS library, you get down to work. The books have already been packed in identical boxes which all have the same weight. You want to stack these boxes on wooden pallets in case the storage room gets flooded, but you have a small problem: you don't know how many boxes you can stack before the pallets collapse under the weight. We'll call the maximum number of boxes that can rest on a pallet the *box limit*.

You could methodically place one box on a pallet, then a second, a third, etc., until the pallet breaks but that seems very time consuming (and leads to a very boring contest problem). But if you have one or more pallets to experiment with you might be able to determine the box limit quicker. For example, suppose because of the size of the boxes and the height of the storage room ceiling the maximum number of boxes in any stack is 3. With just one pallet you could first try one box. If the pallet collapses, well, you need to go out and get some stronger pallets. If the pallet holds then you could try a second box. If the pallet collapses now you know the box limit is 1; otherwise, you try a third box and that will let you know whether the box limit is 2 (if the pallet collapses) or 3 (if the pallet doesn't collapse). This approach requires at most three different experiments. However, if you had two pallets you could determine the box limit with at most only two experiments: first you try two boxes on the first pallet; if it holds, you try a third which will let you know whether the box limit is 2 or 3. If the first pallet collapses in the first experiment, you haul out the second pallet and place a single box on it. The result of that experiment tells you if the box limit is 1 or 0.

You are not exactly sure about the height of the storage room (which dictates the maximum possible boxes that could be stacked) or how many pallets you have at your disposal to experiment with. What you would like to know is when given this information, what is the minimum worst-case number of experiments you need to run given an optimal strategy. Too bad you didn't know about the CS Library earlier – maybe there was something in one of the books that would help you now.

Input

Input consists of a single line containing two positive integers n and m ($n \leq 5\,000, m \leq 20$), where n indicates the maximum number of boxes that can be stacked (regardless of whether a pallet would be collapsed by them or not) and m is the number of available pallets to experiment with.

Output

Output the minimum worst-case number of experiments that the optimal strategy would require followed by the number of boxes to use in the first experiment. If there is a range of boxes that can be used in the first experiment display the minimum and maximum values of this range separated by a hyphen; if there is only one such number simply output that number.

Example

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

Problem L. AI Jeopardy

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

The robot revolution is finally here, albeit not quite in the highly explosive way envisioned in various science fiction books and movies. It seems that, perhaps due to a small typo in the AI source code, the robots are not taking our lives but instead taking our livelihoods. One early job market fatality in this revolution was the (somewhat niche) occupation as jeopardy player: already in 2011 the *Watson* computer defeated two legendary but inferior human jeopardy champions.

Nowadays, more and more of Jeopardy's *viewers* are AIs themselves and as such the show is considering having categories on topics that are more popular with this new number-crunching viewer base. Focus group testing has revealed that AIs are particularly fond of the "Binomial Coefficients" category. The premise of this category is that the answer that contestants get is some positive integer X, and the contestants must respond with a question of the form "What is n choose k?" (and this is a correct response if the binomial coefficient n choose k equals X).

Write an AI to play this new Jeopardy category. If there are several different possible solutions for n and k, the AI should choose the most elegant solution, having the smallest value of n, and of those with the smallest n it should choose the one with the smallest value of k.

Input

Input consists of a single integer X ($1 \leq X \leq 10^{100}$).

Output

Output two non-negative integers n and k such that the binomial coefficient n choose k equals X, with ties between multiple solutions broken as explained above.

Example

Input	Output
10	5 2
2020	2020 1
1	0 0

Problem M. Bling

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

Trapped at home in quarantine, Johan tries to keep madness at bay and fend off isolation by playing Critter Junction, a social simulation video game. One of the main aspects of the game is collecting and harvesting various types of resources, in order to gain Bling, the currency of the game. Johan specializes in running the fruit% category of the game, in which the goal is to obtain the maximum amount of Bling in 40 days using only fruits and no other resources.

Each fruit can be sold for 100 Bling, or planted to become a fruit tree (but not both). Every three days, starting on the third day after it was planted, a fruit tree yields three new fruits.

There are also some more exotic fruits that can be bought from the neighboring village. Once per day, the player can travel to the neighboring village and pay 400 Bling to buy a single exotic fruit which you can then plant or sell already on the same day. Analogously to normal fruits, these exotic fruits can be planted into exotic fruit trees which yield three exotic fruits every three days. Each exotic fruit can be sold for 500 Bling.

Any number of fruits/exotic fruits can be harvested, sold and planted during a day (subject to availability of course, e.g. it is not possible to sell more fruits than you actually have), but at most a single exotic fruit can be bought. These activities can be done in any order, so it is for instance possible within the same day to first harvest a few fruits (exotic or not), then sell those fruits for Bling, then use that Bling to buy an exotic fruit, and then either plant or sell that exotic fruit.

Given the current state of Johan's fruit farm, what is the maximum amount of Bling he can achieve in the remaining time?

Input

The input consists of a single line containing six integers d, b, f, t_0, t_1 and t_2 ($1 \leq d \leq 40$, $0 \leq b \leq 500$, and $0 \leq f, t_0, t_1, t_2 \leq 100$), where:

- d is the number of remaining days Johan has
- b is the current amount of Bling Johan has.
- f is the current number of fruits Johan has.
- t_i is the number of fruit trees Johan has that will yield crop i days from today (for $0 \leq i \leq 2$).

Johan currently does not have any exotic fruits or exotic fruit trees.

Output

Output a single integer: the maximum amount of Bling Johan can have after playing d days.

Example

Input	Output
4 0 1 0 0 0	300
5 0 1 0 1 0	1900
6 0 1 1 0 0	2300
10 399 0 0 0 0	399
1 400 0 0 0 0	500