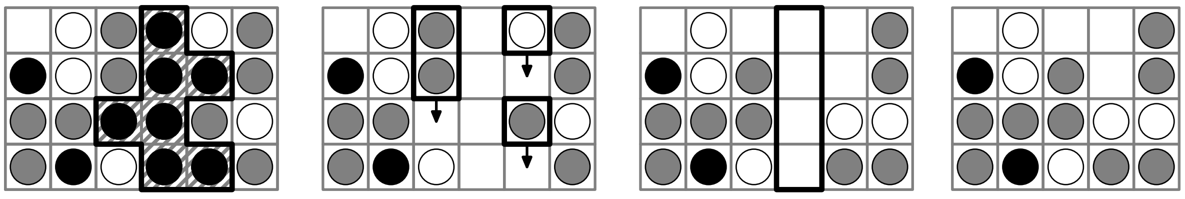
Problem A. Addictive Bubbles

Input file:	`addictive.in`
Output file:	`addictive.out`

Alice is a mobile game developer. She writes a new port of the Chain Shot! game (also known as SameGame, Jawbreaker, Bubble Shot, etc) called Addictive Bubbles.

The game is played on a rectangular board filled with color bubbles. On each turn player selects a group of adjacent bubbles of the same color. Selected bubbles are removed from the board. Bubbles that are no longer supported fall down. If there are empty columns, they are removed.



Selected group Not supported bubbles Empty columns Final state

The number of points scored by the move is a square of the number of bubbles in the selected group. For the sample turn shown on the figure, 49 points are scored.

Turns are repeated until the board is empty. The total number of points is the sum of points scored on each turn.

The blueprint of the level consists of board dimensions and the number of bubbles of each color.

Your task is to help Alice write a bonus level generator. Given the blueprint, generator must produce a level that allows a skillful player to score the maximum possible number of points compared to all levels with the same blueprint.

Input

The input file contains the blueprint.

The first line of the input file contains three positive integers h, w and c — number of rows and columns of the board, and the number of colors ($1 \le h, w \le 10$; $1 \le c \le 9$).

The second line of the input file contains c positive integer numbers — the number of bubbles of each color. The total number of bubbles is $h \cdot w$.

Output

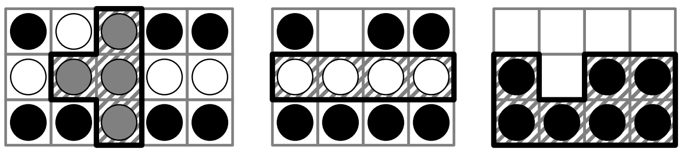
Output the designed level — a $h \times w$ matrix of characters. The bubbles of the first color must be denoted by '`1`', the second color — by '`2`', etc.

Sample input and output

addictive.in	addictive.out

`3 5 3`
`4 4 7`
	`31233`
	`12211`
	`33233`

A sequence of turns, yielding the maximum possible number of points — 81:



16 points 16 points 49 points

Problem B. Blind Problem Solving

Input file:	`standard input`
Output file:	`standard output`

This is an interactive problem. You are to solve a subset sum problem, a variant of a knapsack problem. You are given n items, each weighing w_i grams, and a knapsack with the maximum capacity of c grams. You are to find a subset of these items whose total weight does not exceed c and is maximal possible.

Easy? But you have to do it blindly!

At the beginning you know only two numbers n and c — the number of items and the maximal capacity of a knapsack. There are positive weights w_i and two bit strings inside the jury program: A and B, each having n bits. Each bit string represents a candidate solution to the problem: if the i-th bit is set to 1, then the i-th item is a part of the solution. A stores your previously selected candidate solution. B is a candidate solution that is proposed to you on each turn by flipping a random bit in A. You can either accept or decline this proposal. Your task is to stop when B encodes the optimal solution.

Initially, A is initialized with some value that is fixed for each test case but is not known to you.

At each turn the value of A is copied to B, and then a bit at a uniformly random position of B is flipped. You are given the weight of B which is equal to $\sum_{i=1}^{n} w_i \cdot B(i)$, where $B(i)$ is the value of i-th bit in B.

As a reply, your program is allowed to perform one of the following three actions:

- `stop` — this is the final action. This means that B encodes the optimal solution. Your program must exit after performing this action.
- `accept` — the value of B is copied to A.
- `decline` — the value of B is thrown out.

After each non-final action next turn starts. You are allowed to perform at most 1000 actions.

Interaction protocol

Interaction starts with your program reading three integer numbers — the values of n, c and the weight of B for the first turn from the first line of the standard input. Then your program must write its action to the standard output, wait for the jury program to write the weight of the new value of B to the standard input and so on.

Your program must exit after writing the last action (`stop`) to the standard output. Your program must write end-of-line sequence and flush the standard output after each action, including the last one.

It is guaranteed that a jury program chooses the bits to flip randomly, i. e. using a random number generation algorithm with an initial seed fixed for each test case.

Input

The first line of the standard input contains n ($1 \le n \le 20$), c ($0 \le c \le 10^9$) and the weight of B. It is guaranteed that $1 \le w_i \le 10^8$ for any i.

Each of the following lines will contain one integer number — the weight of B at the corresponding turn.

Output

The standard output consists of the actions your program performs. Each action is represented with a single line containing a single word: `stop`, `accept`, or `decline`.

Sample input and output

standard input	standard output
2 5 3	decline
2	accept
5	stop

Problem C. Caravan Robbers

Input file:	`caravan.in`
Output file:	`caravan.out`

Long long ago in a far far away land there were two great cities and The Great Caravan Road between them. Many robber gangs "worked" on that road.

By an old custom the i-th band robbed all merchants that dared to travel between a_i and b_i miles of The Great Caravan Road. The custom was old, but a clever one, as there were no two distinct i and j such that $a_i \le a_j$ and $b_j \le b_i$. Still when intervals controlled by two gangs intersected, bloody fights erupted occasionally. Gang leaders decided to end those wars. They decided to assign each gang a new interval such that all new intervals do not intersect (to avoid bloodshed), for each gang their new interval is subinterval of the old one (to respect the old custom), and all new intervals are of equal length (to keep things fair).

You are hired to compute the maximal possible length of an interval that each gang would control after redistribution.

Input

The first line contains n ($1 \le n \le 100\,000$) — the number of gangs. Each of the next n lines contains information about one of the gangs — two integer numbers a_i and b_i ($0 \le a_i < b_i \le 1\,000\,000$). Data provided in the input file conforms to the conditions laid out in the problem statement.

Output

Output the maximal possible length of an interval in miles as an irreducible fraction p/q.

Sample input and output

caravan.in	caravan.out

3
2 6
1 4
8 12
	5/2

In the above example, one possible set of new intervals that each gang would control after redistribution is given below.

- The first gang would control an interval between $7/2 = 3.5$ and $12/2 = 6$ miles which has length of $5/2$ and is a subinterval of its original $(2, 6)$.
- The second gang would control an interval between $2/2 = 1$ and $7/2 = 3.5$ miles which has length of $5/2$ and is a subinterval of its original $(1, 4)$.
- The third gang would control an interval between $16/2 = 8$ and $21/2 = 10.5$ miles which has length of $5/2$ and is a subinterval of its original $(8, 12)$.

Problem D. Disjoint Regular Expressions

Input file:	`disjoint.in`
Output file:	`disjoint.out`

Mark is developing new social network *Facepalm* for inhabitants of Phobos and Deimos. His recent task is to add information about home asteroid of the owner to each account. Of course, each account owner could enter such information, but Mark decided that it would be more convenient if some default value was suggested to the user at logon. He investigated the situation and found out that home asteroid of a user can be found by analyzing his last name.

Last name of each user of Facepalm is a non-empty word consisting of lowercase letters of the English alphabet. Users from Phobos have their last names matching regular expression P while users from Deimos have their last names matching regular expression D.

However, the problem is that some last names can match both expressions. Two expressions are called *disjoint* if there is no such non-empty string s that matches both expressions. Mark believes that expressions P and D are disjoint. However he needs your help to check it.

You are given two regular expressions P and D. Check whether they are disjoint, and if they are not, find the shortest non-empty string s that matches both of them. If there are several shortest common strings, you can find any one.

Note

Let us define regular expressions and matching strings to them formally.

- A single lowercase letter c is a regular expression, it matches only a string consisting of a single letter c.
- Alternation: if P and Q are regular expressions then $(P|Q)$ is a regular expression, a string α matches it if α matches P or α matches Q.
- Concatenation: if P and Q are regular expressions then (PQ) is a regular expression, a string α matches it if $\alpha = \beta\gamma$, β matches P and γ matches Q.
- Kleene star: if P is regular expression then $(P*)$ is a regular expression, a string α matches it if α can be represented as a concatenation of zero or more strings $\alpha_1\alpha_2\ldots\alpha_k$ where all α_i match P. An empty string always matches Kleene star.

Parentheses can be omitted, in this case Kleene star has the highest priority, then concatenation and then alternation. For example, “`abc*|de`” means “`(ab(c*))|(de)`”.

Input

The input file contains two lines. The first line contains regular expression P. The second line contains regular expression D. Each expression contains from 1 to 100 characters.

Output

If Mark's guess is correct and the two expressions are indeed disjoint, print “`Correct`” at the first line of the output file. If they are not, print “`Wrong`” at the first line of the output file. In this case the second line must contain any shortest non-empty string that matches both expressions.

Sample input and output

`disjoint.in`	`disjoint.out`

`a(ab)*b`
`a(a|b)*ab`
	`Correct`

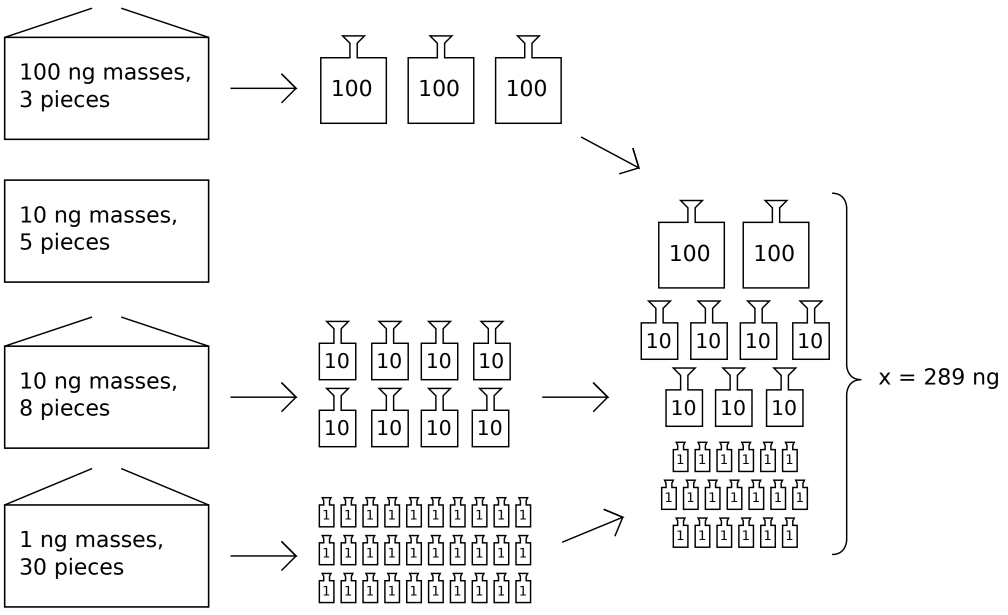
`a(ab)*a`
`a(a|b)*ba`
	`Wrong`
	`aaba`

Problem E. Exact Measurement

Input file:	`exact.in`
Output file:	`exact.out`

Peter is working in a secret chemical laboratory. For his new experiment he needs to measure exactly x nanograms of a secret reagent. He has a balance and several standard masses, and his goal is to choose a set of standard masses with total sum equal to x ng.

Standard masses come in n sealed boxes. The i-th box contains q_i identical masses of 10^{k_i} ng. Peter wants to open the minimal number of boxes to take a set of masses with the sum of their weights of x ng.



Input

The first line of the input file contains two integer numbers x and n ($1 \le x \le 10^{18}$, $1 \le n \le 10^5$). The next n lines contain pairs of numbers k_i and q_i ($0 \le k_i \le 18$, $1 \le q_i \cdot 10^{k_i} \le 10^{18}$).

Output

On the first line output the minimal number of boxes that should be opened. On the second line output the numbers of these boxes in any order. Boxes are numbered in the order they appear in the input file starting from 1. If it is impossible to measure exactly x ng, output a single line with -1.

Sample input and output

exact.in	exact.out

289 4
2 3
1 5
1 8
0 30
	3
	1 3 4

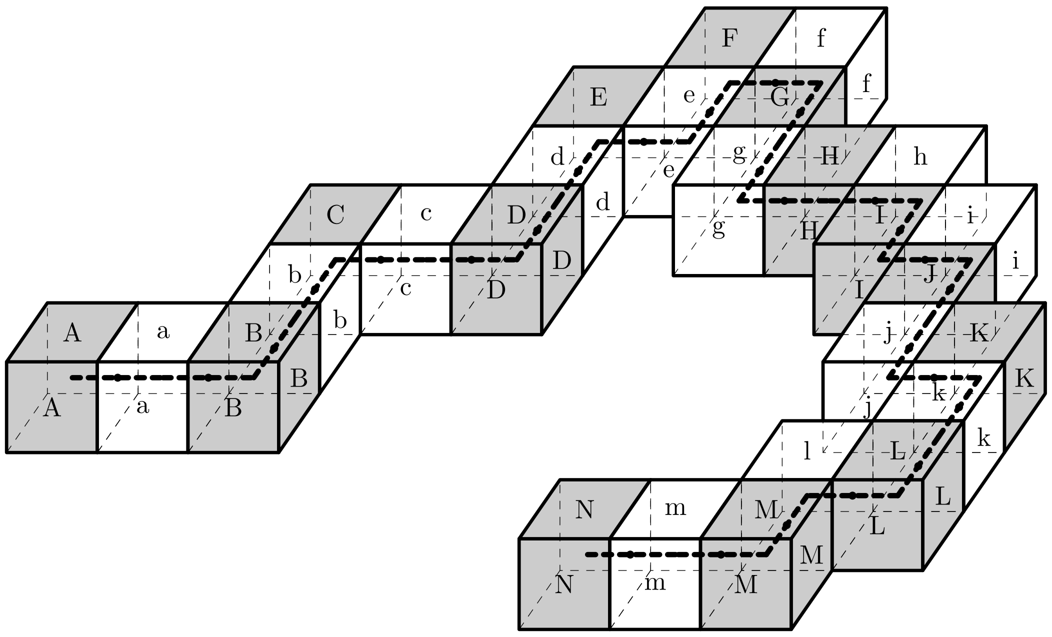
300 4
2 3
1 5
1 7
0 30
	1
	1

201 1 2 3	-1

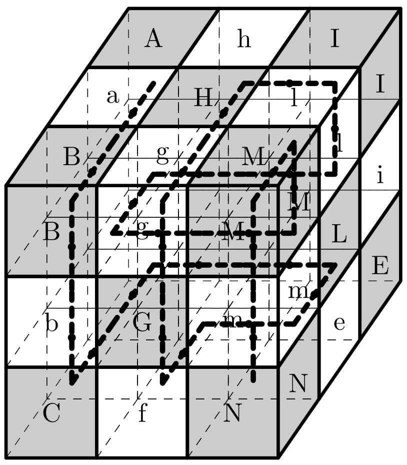
Problem F. Folding Snake Cube

Input file:	`folding.in`
Output file:	`folding.out`

Snake Cube is a puzzle that consists of 27 wooden cubelets that are connected by an elastic band running through them. The band starts in the center of the first cubelet, exits through the center of one of its faces that is adjacent to the face of the second cubelet. Then the band enters through the center of the adjacent face into the second cubelet, exits the second cubelet through the center of some other face, enters the third cubelet and so on until the last cubelet where the band ends. This configuration allows adjacent cubelets to freely turn around the center of their shared faces. Moreover, the elasticity of the band allows to stretch it during intermediate steps and to fold the puzzle in a ways that would be otherwise impossible with rigid connections between cubelets. The task is to fold it into a $3 \times 3 \times 3$ cube.



Cubelets in the Snake Cube puzzle are typically colored into two alternating colors. For the purpose of this problem all cubelets are marked by alternating uppercase and lowercase letters. The first cubelet is marked by 'A', the second one by 'a', the third one by 'B', the fourth one by 'b', and so on. The last 27th cubelet is marked by 'N'. The order of marking follows the path of the band through the cubelets.



Initially the puzzle is laid out on a flat surface and its layout is given the input file. Your task is to find a way to fold it into a $3 \times 3 \times 3$ cube and describe the resulting layout in the output file.

Input

The input file consists of 15 lines with 15 characters on each line that represent flat layout of the puzzle. Dot characters ('.') represent empty space. Uppercase letters from 'A' to 'N' and lowercase letters from 'a' to 'm' represent marked cubelets with the band running through AaBb...N as explained in the problem statement.

The input file represents a valid layout. In a valid layout:

- each marked cubelet appears exactly once;
- cubelets that are adjacent in the order that elastic band runs through them are also adjacent in the layout;
- a face that the band exists a cubelet from is adjacent to the face of the next cubelet that the band enters into.

The puzzle given in the input file is solvable and can be folded into a $3 \times 3 \times 3$ cube.

Output

Write to the output file 3 lines with 11 characters on each line (3 groups of 3 characters separated by single spaces) that represent the puzzle in the input file folded into a $3 \times 3 \times 3$ cube. The output file corresponds to 3 consecutive 3×3 slices of the resulting layout.

The output file layout has to be valid in the same sense as in the input file.

Sample input and output

Sample input and output correspond to the layouts that are shown on the pictures in the problem statement.

folding.in	folding.out
....	AhI jJi DdE
....	aHl KkL cFe
......Ff.......	BgM bGm CfN
.....EeG.......	
.....d.gHh.....	
...CcD...Ii....	
...b......J....	
.AaB......jK...	
...........k...	
..........lL...	
........NmM....	
....	
....	
....	
....	

Problem G. Great Deceiver

Input file:	`great.in`
Output file:	`great.out`

Once upon a time Baron Munchhausen traveled to the Moon. After that he often tells interesting stories about the Selenites. Recently Baron told us about their numeric system. They use a notation with negative radix!

Negative radix is quite hard for Humans, and even for Munchhausen. So, Baron did a trick to help himself on the Moon. He remembered all the numbers between 0 and n inclusively, which have the same notation for both Selenites' negative radix $-k$ and a more convenient positive radix k.

Munchhausen claims that he did that easily. But, you know, Baron can exaggerate a little. To catch him, you have to count how many numbers he must have remembered.

Note: the k-radix notation of a number x is a sequence of integers $a_0, a_1, ..., a_p$ such that $0 \le a_i < |k|$ and $\sum_{i=0}^{p} a_i \cdot k^i = x$.

Input

The single line of the input file contains two integer numbers n and k ($1 \le n \le 10^{15}$, $2 \le k \le 1\,000$).

Output

Output the number of numbers Baron Munchhausen must have remembered during his stay on the Moon.

Sample input and output

great.in	great.out
21 3	9
21 2	8

In the first sample, Baron must have remembered numbers 0, 1, 2, 9, 10, 11, 18, 19 and 20.

For example, 19 has the same notation for both radix 3 and radix -3: $19 = 201_3 = 201_{-3}$, while 7 has not: $7 = 21_3 = 111_{-3}$.

In the second sample, the corresponding numbers are 0, 1, 4, 5, 16, 17, 20 and 21.

Problem H. Hyperdrome

Input file:	`hyperdrome.in`
Output file:	`hyperdrome.out`

Hypergnome planet is famous for its Great Universal Games between gnomes — the Games between gnomes from each part of the galaxy in various disciplines.

The most popular discipline in the Games is the Hyperdrome discipline. The rules are the follows: one string of length n is given to all gnomes. The gnomes shall find, as fast as they can, the total number of Hyperdrome substrings — such strings that characters inside the string can be rearranged to get a palindrome.

Substring is defined as a sequence of characters from position i to position j inclusive, where $1 \le i \le j \le n$. Substrings with different pairs of positions (i, j) are considered different regardless of their contents.

Palindrome is defined as a string $x_1x_2...x_l$, where $x_i = x_{l-i+1}$ for all $1 \le i \le l$.

Judges choose a string and your task is to help them find the answer.

The gnome alphabet consists of lowercase and uppercase English letters — 'a'–'z' and 'A'–'Z' where letters in different case are considered to be different letters.

Input

The first line of the input file contains a single integer n ($1 \le n \le 3 \cdot 10^5$).

The second line of the input file contains the string for Hyperdrome discipline — n lowercase or uppercase English letters.

Output

Output the answer for the Hyperdrome discipline — the number of Hyperdrome substrings in the input string.

Sample input and output

hyperdrome.in	hyperdrome.out
3 aaa	6
7 abadaba	12
3 aAA	5

In the first example there are 6 Hyperdrome substrings — $(1,1)$, $(1,2)$, $(1,3)$, $(2,2)$, $(2,3)$, $(3,3)$.

In the second example there are 12 Hyperdrome substrings — $(1,1)$, $(2,2)$, $(3,3)$, $(4,4)$, $(5,5)$, $(6,6)$, $(7,7)$, $(1,3)$, $(3,5)$, $(5,7)$, $(2,6)$, $(1,7)$.

In the third example there are 5 Hyperdrome substrings — $(1,1)$, $(1,3)$, $(2,2)$, $(2,3)$, $(3,3)$. Note, that a Hyperdrome substring $(1,3)$ is "`aAA`". It is not a palindrome itself, but its characters can be rearranged to get a palindrome "`AaA`"

Problem I. Identification of Protein

Input file:	`identification.in`
Output file:	`identification.out`

A protein is a sequence of amino acids. One of the ways to sequence (i.e. to read) a protein is to use *Tandem mass spectrometry* (usually abbreviated as MS^2).

In this problem the following toy situation is considered:

- A protein is a sequence of at most 400 amino acids.
- The are only two types of amino acids, P and Q.
- Mass of P is 97.05276 daltons, mass of Q is 128.05858 daltons.

The result of an ideal measurement via MS^2 is a set of real numbers called *peaks*, which contains the mass of each prefix and each suffix of the protein (including the entire protein), and no other numbers.

But it's never ideal in practice. You're given the results of MS^2 experiment for an unknown protein. The following conditions hold:

- masses of some prefixes and/or some suffixes might be absent in the set.
- some peaks that do not correspond to any prefix/suffix mass might be included in the set (they are called *noise peaks*).
- all peaks are positive numbers.
- the largest peak in the set is equal to the mass of the entire protein.
- all peaks that correspond to prefix/suffix masses are exact measurements.

Reconstruct the protein that has the mass equal to the largest peak and minimizes the number of noise peaks (and thus maximizes the number of peaks that correspond to its suffixes or prefixes).

Input

The first line contains integer n ($1 \le n \le 100\,000$), the number of peaks in the given MS^2 experiment results.

Each of the next n lines contains a positive real number p_i, the mass of the i-th peak.

All of above conditions are guaranteed to be satisfied. All peaks are distinct. Each peak has no more than 5 digits after a decimal point.

Output

Output a string of up to 400 characters consisting only of '`P`' and '`Q`' characters — the reconstructed protein. If there are multiple answers that minimize the number of noise peaks, output any one of them.

Sample input and output

`identification.in`	`identification.out`

`6`
`225.11134`
`353.16992`
`353.16991`
`291.15828`
`97.05276`
`128.05858`
	`PQQ`

In the given example, the protein PQQ (or QQP) explains all peaks except two: 291.15828 and 353.16991.

Problem J. Jumping Around

Input file:	`jumping.in`
Output file:	`jumping.out`

John, Jill and Jeremy are planning their vacation trip. They would like to visit all planets in their planetary system. They are planning to use *Telejump* teleport system which was recently installed at all planets. There are n planets in the planetary system, numbered from 0 to $n-1$. John, Jill and Jeremy are planning to start their journey from planet 0 and can finish it at any planet.

Telejump uses three types of tickets for their system. Ticket of the first type allows to travel from planet x to planet $x+1$ (when $x+1 \le n-1$) or to planet $x-1$ (when $x-1 \ge 0$). Ticket of the second type allows to travel from planet x to planet $x+2$ (when $x+2 \le n-1$) or to planet $x-2$ (when $x-2 \ge 0$). Finally, ticket of the third type allows to travel from planet x to planet $x+3$ (when $x+3 \le n-1$) or to planet $x-3$ (when $x-3 \ge 0$).

Friends have bought a tickets of the first type, b tickets of the second type and c tickets of the third type. Tickets are quite expensive, so they have bought the minimal number of tickets they need to visit all planets: $a+b+c=n-1$. However, all three of friends collect used *Telejump* tickets, so they have bought at least 3 tickets of each type (yes, you can deduce from these facts that $n \ge 10$).

Now they would like to plan their trip.

Help John, Jill and Jeremy to choose in which order they should visit planets, so that they could visit each planet by using their tickets.

Input

The input file contains several test cases. The first line of the input file contains t — the number of test cases ($1 \le t \le 20$).

Each of the following t lines contains three integers each: a_i, b_i and c_i ($3 \le a_i, b_i, c_i \le 5000$). For such test case n_i is equal to $a_i + b_i + c_i + 1$.

Output

Output one line for each test case in the input. Each line must contain n_i integers separated by spaces: planet numbers in the order friends should visit them to use their tickets.

If there are several solutions, output any one. It is guaranteed that for each test case from the input file solution always exists.

Sample input and output

jumping.in	jumping.out

`2`
`3 3 3`
`3 4 3`
	`0 3 1 2 5 4 6 9 7 8`
	`0 3 1 2 5 4 6 9 7 8 10`

In the first example above there are 3 tickets of each type, therefore there are 10 planets numbered from 0 to 9. Friends start at planet 0. They use tickets of the first type for their jumps $1 \to 2$, $5 \to 4$, and $7 \to 8$, tickets of the second type for jumps $3 \to 1$, $4 \to 6$, and $9 \to 7$, and tickets of the third type for jumps $0 \to 3$, $2 \to 5$, and $6 \to 9$.

Problem K. Kingdom Reunion

Input file:	`kingdom.in`
Output file:	`kingdom.out`

Kalvin John Ian Helvik IV and Karl James Ingram Helvik III are second cousins and kings of neighbouring countries, Aastria and Abstria. One hundred years ago their countries constituted a single country, but the old king Helvik died and left his country to his twin sons, Ian and Ingram. Nobody knew what to do, until it was suggested to split the country into two equal parts. This was an overwhelmingly complex task, which the wisemen tried to cope with by calling the best minds from the future to solve.

Unfortunately, five attempts to solve the problem had failed, and the bloody civil war broke out. It lasted for seven years and ended with the NEERC (Northeastern Enormously Ragged Combat). As the Final Ordinance said, there would be two new countries in the place of the old one, one for Ian and one for Ingram. But these two countries had unequal areas, so it took only three years for war to start again.

After ninety years of war, all resources of two countries were exhausted. At the end of the year both kings had realized that they would not survive the next year if the war continued, so they simultaneously sent envoys with offers of peace. They had decided to unite their countries back and live in peace. The united land of Aastria and Abstria was named Aabstria.

In the manuscripts of the old ages you had found several descriptions of boundaries of the countries. Each description is a sequence of locations of boundary monuments, which are listed in clockwise or counterclockwise order following the boundary. However, you suddenly realized that in different sources locations of boundary monuments differ. In some of them, the boundary does not even form a polygon. You decided to do an investigation to discover what the actual boundaries of the countries were. Given locations of the boundary monuments for each of three countries the following statements should hold:

- For each country, the polyline formed by the boundary monuments should be a *polygon*. A polyline is considered to be a *polygon* if it has at least three points, no self-intersections, self-touches or holes, and has a non-zero area.
- The interiors of Aastria and Abstria polygons should not intersect.
- The union of Aastria and Abstria should be precisely equal to Aabstria.

Your task is to write a program that checks if these facts are true.

Input

The first line of the input file contains the number of vertices in the boundary of Aastria, n_a ($1 \le n_a \le 10\,000$). The next n_a lines contain two integers each — the coordinates of Aastria boundary monuments in clockwise or counterclockwise order.

After that, descriptions of Abstria and Aabstria are given in the same format as above.

The coordinates of all monuments do not exceed 10^5 by their absolute values. It is guaranteed that two boundary monuments can coincide only if they belong to different countries.

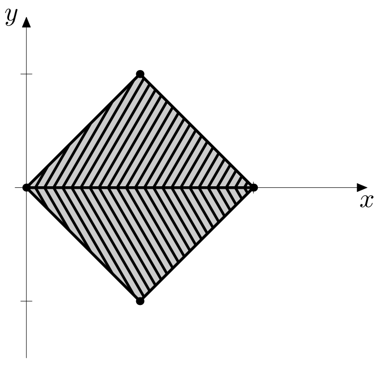
Output

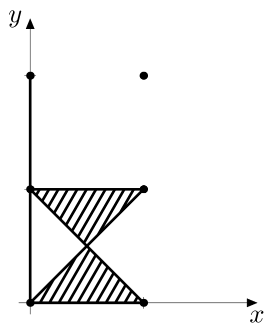
Output a single line with one the following strings (without quotes):

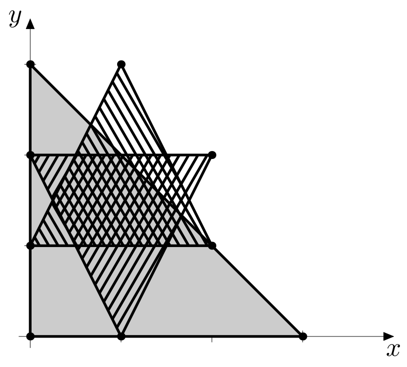
- if the boundary monuments of Aastria do not form a polygon, output "`Aastria is not a polygon`";
- otherwise, if the boundary monuments of Abstria do not form a polygon, output "`Abstria is not a polygon`";
- otherwise, if the boundary monuments of Aabstria do not form a polygon, output "`Aabstria is not a polygon`";
- otherwise, if interiors of Aastria and Abstria intersect, output "`Aastria and Abstria intersect`";
- otherwise, if the union of Aastria and Abstria is not equal to Aabstria, output "`The union of Aastria and Abstria is not equal to Aabstria`";
- otherwise, output "`OK`".

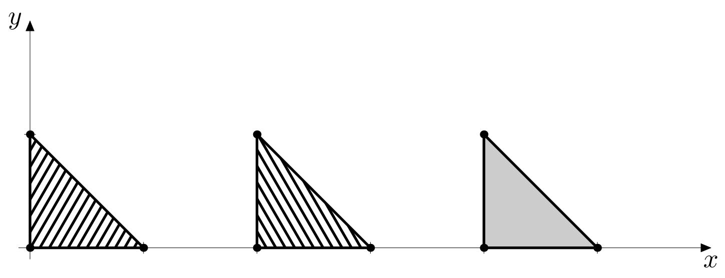
Sample input and output

kingdom.in	kingdom.out

3
0 0
2 0
1 1
3
0 0
2 0
1 -1
4
0 0
1 1
2 0
1 -1
	OK
	

4
0 0
1 1
0 1
1 0
3
0 0
0 1
0 2
1
1 2
	Aastria is not a polygon
	

3
0 2
1 0
2 2
3
0 1
2 1
1 3
3
0 0
3 0
0 3
	Aastria and Abstria intersect
	

3
0 0
1 0
0 1
3
2 0
3 0
2 1
3
4 0
5 0
4 1
	The union of Aastria and Abstria is not equal to Aabstria
	

Problem L. Labyrinth of the Minotaur

Input file:	`labyrinth.in`
Output file:	`labyrinth.out`

The Minotaur is a half-bull half-man creature living in the Cretan Labyrinth. He terrorizes the whole Crete, especially the city of Minos. Every year seven young boys and girls are sent to the Labyrinth to please the Minotaur. After each sacrifice the Minotaur sleeps for a while.

Theseus, brave Greek hero half-god half-man, just came to Minos. The people of Minos ask him to kill the Minotaur. Unfortunately, the Minotaur is not an easy target to kill. Theseus doubts his ability to kill even sleeping Minotaur. So he decided to block the Minotaur inside his own Labyrinth.

The Labyrinth has a rectangular shape divided into square cells of equal size. Each cell is either empty of blocked. Blocked cells are impassable even for the Minotaur. The entrance to the Labyrinth is located in one corner of the Labyrinth, while the Minotaur's lair is located in the opposite corner.

Theseus has only one chance to block the Minotaur — while he is asleep after sacrifice quickly build a square obstacle that blocks some of the Labyrinth's cells. The cells that the obstacle is built on must be empty. The Minotaur is blocked if there is no way from his lair to the entrance of the Labyrinth. Certainly, the obstacle cannot block the Minotaur's lair cell (you cannot build something a top of the Minotaur, even on a sleeping one), as well as the entrance cell (Theseus must not block the Labyrinth completely).

You have to calculate the minimum possible size of the square obstacle that is able to block the Minotaur.

Input

The first line of the input file contains a pair of positive integer numbers w and h — the width and the height of the Labyrinth ($2 \le w, h \le 1500$).

The following h lines contain map of the Labyrinth. Each of them has length of w characters. Empty cells are denoted by dots (`.`) and blocked cells — by number signs (`#`).

The entrance is located in the upper-left corner (cell $(1, 1)$) and the Minotaur's lair is located in the bottom-right corner (cell (w, h)). Both of these cells are empty and there is at least one way from the entrance to the Minotaur's lair.

Output

Output three integer numbers l, x and y – the length of side of the minimum possible square obstacle that is able to block the Minotaur inside the Labyrinth, and the coordinates of its upper left cell. If there are multiple pairs of possible coordinates, output any of them. The obstacle must not contain any blocked cells, as well as the Labyrinth entrance or the Minotaur's lair. If it is not possible to build a square obstacle that blocks the Minotaur, output a single word "`Impossible`".

Sample input and output

labyrinth.in	labyrinth.out

`11 6`
`......#####`
`.#.#...#..#`
`.#.#.......`
`.......###.`
`#####.###..`
`#####......`
	`2 6 3`

`3 3`
`...`
`.#.`
`...`
	`Impossible`