Problem A. Bubbles *(Division 2)*

Input file:	`bubbles.in`
Output file:	`bubbles.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Today is the Hexadecimal's birthday!

She decided to entertain herself in an extremely awkward way: blowing bubbles. A bubble is a circle on the plane.

Let us start with choosing a point on Ox axis and lets name this point a *blowation* center. Yep, the center of all the bubbles will be a single point which is the blowation center.

After choosing the blowation center, we can start blowing bubbles. The i-th bubble is blowing until it either touches the point (x_i, y_i) or touches the surface of any other bubble which is already blown. In the case one bubble touches another one, a myriad of phantasmagorically illuminating splashes will appear, and this is absolutely prohibited.

So, how many orders of blowing exist such that the absolutely prohibited thing does not happen?

Input

The number of bubbles n is given in the first line of input ($1 \le n \le 1000$).

Each of the following n lines contains a space-separated pair of integers. The i-th pair contains coordinates of the i-th point (x_i, y_i) ($1 \le x_i, y_i \le 100$).

Output

Print the only integer, the number of allowed orders of blowing.

Examples

bubbles.in	bubbles.out
2 3 3 7 3	2
3 1 1 2 4 9 2	4

Explanations

In the first example, it is possible to choose, for example, $x = 0$ for blowation point and blow bubbles in the order $(1, 2)$, or choose, for example, $x = 10$ for blowation point and blow bubbles in the order $(2, 1)$.

In the second example, the possible orders are $(1, 2, 3)$, $(2, 1, 3)$, $(2, 3, 1)$ and $(3, 2, 1)$.

Problem B. Drop7 *(Division 2)*

Input file:	`drop7.in`
Output file:	`drop7.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Each workday, Eugene uses the subway to travel between home and work. It is very difficult to use internet inside a subway train, so Eugene plays games on his iPad. His favorite game is Drop7.

It is highly recommended to read the description of game rules carefully even if you already played the game Drop7. Game rules in this problem are slightly different from the original rules.

The game board is a rectangle which has 8 cells in height and 7 cells in width. Each cell of the board can either be empty or contain one *disc* with digit between 1 and 7. Each disc could have zero, one or two protection layers. Discs are affected by gravity force which is directed downwards. If the cell under some disc is empty, this disc will be moved down by gravity.

Each player's move is to throw one disc into one of the seven columns of the game board.

Discs on the board can explode according to the following rules. Disc with digit X *without protection layers* will detonate and then explode if it belongs to horizontal and/or vertical line of exactly X *consecutive* discs. Detonations will only happen at the moment when there are no more discs moving down by gravity. At first, all the discs on the board satisfying the condition described above will simultaneously detonate, then all these discs will simultaneously explode. A disc loses one protection layer when another disc has exploded in a cell adjacent by an edge. Effects of explosions are added together, so a disc loses as many protection layers as is the number of explosions happened in adjacent cells. A disc can not lose more protection layers than it has. So, a disc without any protection layers can not lose them anymore.

After explosion of all detonated discs, some discs can start moving down by gravity. When they all stop, it can lead to the next wave of explosions and fallings. The process continues until the remaining discs stop to explode and fall.

The player starts the game on the first level. After several player moves, the player reaches the next level. When it happens, all discs on the board move one cell up, and seven new discs appear in the bottom row. This event can lead to detonation of some discs which can in turn lead to a chain of explosions. If some disc appears in the highest row, the game stops *immediately*.

So, the player's move can be described as follows:

- The player throws a new disc.
- If there is a disc in the highest row, the game stops.
- While there are detonated discs, the wave of explosions and fallings occurs.
- If the player reached the next level, the new discs appear in the bottom row.
- If there is a disc in the highest row, the game stops.
- While there are detonated discs, the wave of explosions and fallings occurs.

The player scores points for exploded discs depending on the number of the wave on which the disc has exploded. The waves are enumerated starting from one, after each thrown disc the numbering is started from one again, but when the player reaches the next level, the numbering continues. For each disc exploded on the wave i, the player gets $\frac{7i^3+72i^2-73i+36}{6}$ score.

There are two difficulty modes in the game: Normal and Hardcore. On Hardcore difficulty, the player reaches the next level after each 5 moves and gets 17 000 score per each new level. On Normal difficulty, the player needs 30 moves to reach the second level from the first, 29 moves to reach the third level from the second, ..., 6 moves to reach the 26-th level from the 25-th. After reaching the 26-th level, the player needs 5 moves to reach each next level. On Normal, the player gets 7000 score per each new level.

Unfortunately, Eugene was unable to find this game anywhere except AppleStore. So he came to you and asked you to write an emulator of the game Drop7. While you are writing the code, Eugene decided to prepare test data for you. He started a new game, wrote the initial state of the board on the sheet of paper and made some moves, logging all information needed to recover the game process. He wrote which disc he threw and where he threw it on each move, and also he wrote which seven discs appeared in the bottom row each time he reached the next level. If after some move he had reached the next level, he *always* recorded the appeared discs even if it was the last move he did.

You are required to output the final state of the game using the information Eugene wrote.

Input

The first line of input describes the difficulty setting, so there are two possible cases: "`Mode: Normal`" and "`Mode: Hardcore`". The next eight lines describe the initial state of the game board. Each line has exactly seven characters. Character '`.`' describes an empty cell, characters '`1`'–'`7`' describe a disc with corresponding digit without any protection layers, characters '`a`'–'`g`' describe a disc with one protection layer, and characters '`A`'–'`G`' describe a disc with two protection layers. It is guaranteed that on the initial game board, there are no discs in the highest row, no discs that could fall and no discs that could detonate.

Eugene used a very compact notation to write all remaining information, so there is only one line of it, written on the last line of the input. Let us call it the game log. The game log can contain from 1 to 10 000 characters from the set "`1234567abcdefgABCDEFG`". When Eugene made his move, he wrote two characters to the game log: the first character described the disc, the second one was the number of the column (columns are enumerated from left to right) where this disc was thrown. For example, "`b7`" means that Eugene has thrown the disc with digit 2 with one protection layer to the seventh column if we count from the left. When Eugene reached the next level, he wrote seven characters to the log describing the new discs appeared in the bottom row (the first character describes the disc in the leftmost column, the last one describes the disc in the seventh column counting from the left).

Output

Eugene could make mistakes while writing the log. If the log cannot be interpreted as a correct sequence of moves, you need to output an error message. You need to output the message "`Game log is not complete`" if the log is a correct prefix of the game record, or the message "`Error in game log at position` X" if the prefix of length $X - 1$ is a correct prefix of the game record, but the prefix of length X is not.

If the log is correct, you need to output the final state of the game. If Eugene lost the game, you need to output the line "`Game is over!`" first. Then output the score Eugene got, the maximal length of explosion chain during the game and the level reached. Then output the final state of the game board. Adhere to sample output shown below as close as possible.

Examples

drop7.in	drop7.out
Mode: Normal .... 3A..g.. 3314e2e2e2e2e2e2e2e2	Game is over! Score: 60 Longest chain: 2 Level reached: 1 .e..... .e..... .e..... .e..... .e..... .e..... .e..... .e..7..

drop7.in	drop7.out
Mode: Normal A1A1A1A1A1A1A1A1A1	Error in game log at position 17
Mode: Hardcore A1B	Game log is not complete
Mode: Hardcore A1B2C3D4E5	Game log is not complete
Mode: Hardcore 71g167f2b3BBbbbbbc436b232A5Abcdefg55	Score: 64418 Longest chain: 14 Level reached: 3 ....5..

Explanations

In the second example, Eugene loses after eight moves corresponding to the first 16 characters of the game log. When the game ended, there could not be any other events, so the log should have been terminated after 16-th character.

In the third example, the log ends inside a description of a move.

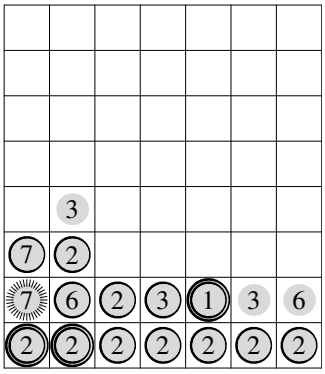
In the fourth example, Eugene reaches the second level after the first five moves, but there is no information about the discs appeared in the bottom row in the game log.

In the fifth example, the first five moves ("71", "g1", "67", "f2", "b3") did not lead to any explosions. After

fifth move, Eugene reached the second level, and the bottom row was filled with new discs ("`BBbbbbb`"). The next four moves ("`c4`", "`36`", "`b2`", "`32`") still did not lead to any explosions. On the tenth move ("`A5`"), Eugene started the chain of 14 explosions by throwing disc with digit 1 into the fifth column counting from the left. Between 6-th and 7-th explosion, Eugene reached the third level (and new discs appeared: "`Abcdefg`"). The sample pictures showing the first 8 explosions and reaching the third level are shown below.

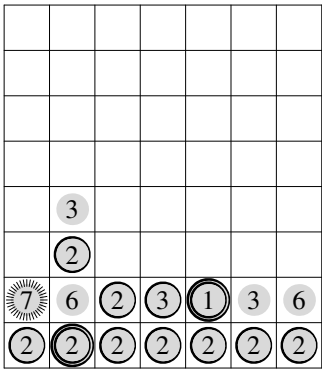
Score: 17000
Level: 2

Chain#1 +7



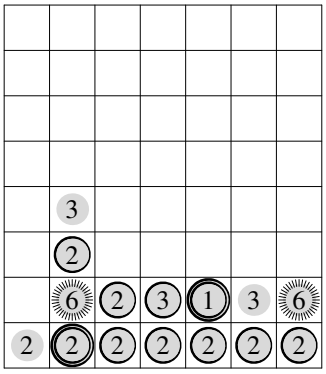
Score: 17007
Level: 2

Chain#2 +39



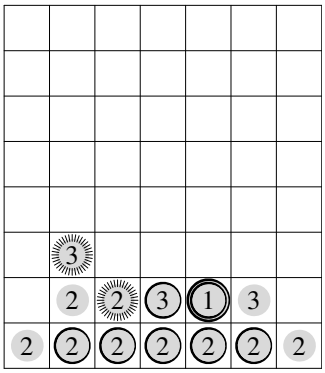
Score: 17046
Level: 2

Chain#3 +2x109



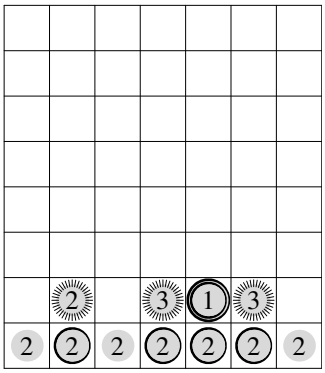
Score: 17264
Level: 2

Chain#4 +2x224



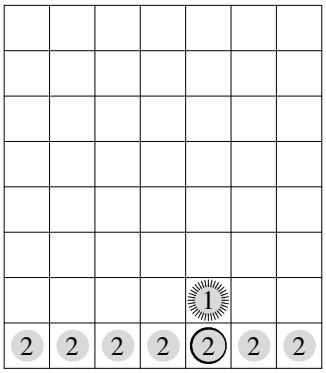
Score: 17712
Level: 2

Chain#5 +3x391



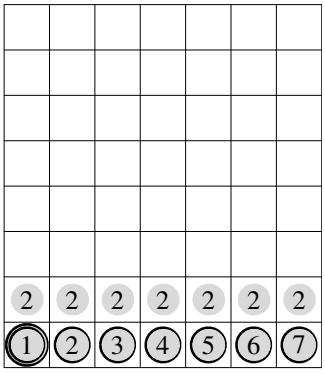
Score: 18885
Level: 2

Chain#6 +617



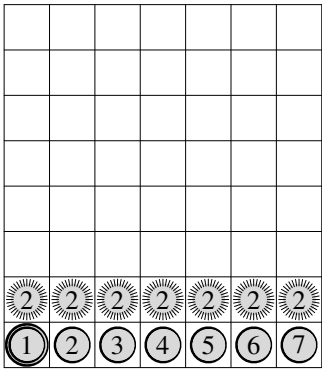
Score: 36502
Level: 3

Level up! +17000



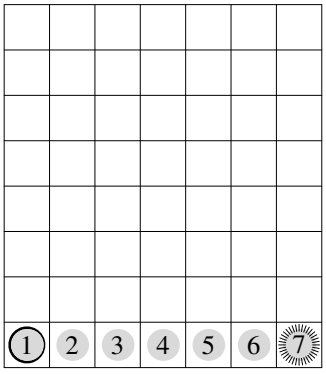
Score: 36502
Level: 3

Chain#7 +7x909



Score: 42865
Level: 3

Chain#8 +1274



Problem C. Eulerian Graphs *(Division 2)*

Input file:	`euler.in`
Output file:	`euler.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

The night before the competition Alice had read the book about graphs and found some interesting definitions in it.

A graph G is a pair of sets: vertices and edges (V, E) where the set of edges E is a subset of the set of unordered pairs of vertices. A cycle in graph (V, E) is a non-empty sequence of edges $(u_1, v_1), \ldots, (u_m, v_m)$ such that $v_i = u_{i+1}$ for all $i < m$ and $v_m = u_1$.

An Eulerian cycle is a cycle in a graph which contains each edge exactly once. A graph is Eulerian if it contains an Eulerian cycle.

Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are called isomorphic if there is a bijective function $f : V_1 \to V_2$ such that for all v and u from V_1, $(v, u) \in E_1$ if and only if $(f(u), f(v)) \in E_2$.

Having understood this definitions, Alice asked a natural question: "How many non-isomorphic Eulerian graphs with n vertices exist?".

Could you help Alice?

Input

The first line of input contains an integer T: the number of test cases ($1 \le T \le 7$). Each of the following T lines contains an integer n ($1 \le n \le 7$).

Output

For each test case, print the number of non-isomorphic graphs on n vertices taken modulo $10^9 + 7$.

Example

euler.in	euler.out
2 3 4	5 11

Problem D. At Least Half *(Division 2)*

Input file:	`halfgcd.in`
Output file:	`halfgcd.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

As you will no doubt be aware, the plans for development of the outlying regions of the Galaxy require the building of a hyperspatial express route through your star system, and regrettably your planet is one of those scheduled for demolition. The process will take slightly less that two of your Earth minutes. Thank you.

—Prostetnic Vogon Jeltz of the Galactic Hyperspace Planning Council

As all the vogons do, Prostetnic Jeltz loves numbers and loves to destroy planets. Today is his lucky day: he is assigned to eliminate some planets for building a hyperspace tunnel instead of them. This is a black day for all living creatures of Squornshellous system not only because of their possible destruction, but also because all planets form a line, creating a solar eclipse on each planet.

But there is still hope for some inhabitants of Squornshellous system. Vogons have special rules about planet destruction. First of all, for the consistency of galactic harmony, they can destroy only a consecutive segment of planets when they are all on one line. Also vogons know that i-th planet has radius r_i. Secondly, because the Big Laser has very complex stucture, it will start only if there exists a set containing at least half of the doomed planets such that the greatest common divisor of all their radii is greater than one.

Can you help Prostetnic Jeltz to destroy as many planets as possible in such strict conditions?

Input

The first line contains one integer N: the number of planets in Squornshellous system ($1 \le N \le 1000$). The next line contains the radii of the planets $r_1, \ldots, r_N$ in the order in which they appear on the line ($1 \le r_i \le 10\,000$).

Output

Output two integers L and R: the indices of the first and the last planet which will be destroyed by vogons. If there is more than one correct answer, you can output any of them. If you can not destroy anything, output two zeroes instead.

Example

halfgcd.in	halfgcd.out
5 1 2 1 2 1	2 5

Problem E. Next Partition *(Division 2)*

Input file:	next-partition.in
Output file:	next-partition.out
Time limit:	2 seconds
Memory limit:	256 mebibytes

Consider all ways of partitioning a positive integer N into K positive integers. A partition will be written as a sequence of numbers in the order from larger to smaller terms. Let us sort partitions in reverse lexicographical order. Find the next partition of N into K integers in this order, or report that it does not exist.

Input

The first line of input contains one integer K ($K \le 10^5$, $K \le N$, $1 \le N \le 10^9$). The second line contains K integers A_i, the elements of the partition ($1 \le A_i \le N$).

Output

If the next partition does not exist print -1. Otherwise print an integer K on the first line of output. In the second line print K integers: summands of the next partition in reverse lexicographical order.

Examples

next-partition.in	next-partition.out
6 4 2 2 2 1 1	6 3 3 3 1 1 1
3 2 2 2	-1

Explanations

In the first example, $N = 12$ and $K = 6$. The first partition in reverse lexicographical order is $12 = 7 + 1 + 1 + 1 + 1 + 1$, the second is $12 = 6 + 2 + 1 + 1 + 1 + 1$, the 7-th is the given partition $12 = 4 + 2 + 2 + 2 + 1 + 1$, the next one is $12 = 3 + 3 + 3 + 1 + 1 + 1$ and the last 11-th partition is $12 = 2 + 2 + 2 + 2 + 2 + 2$.

Problem F. Equation *(Division 2)*

Input file:	`polynoms.in`
Output file:	`polynoms.out`
Time limit:	2 seconds
Memory limit:	256 mebybytes

A polynomial $P(x)$ over the field $\mathbb{F}_2$ is an expression of the form $\sum_{i=0}^{n} c_i \cdot x^i$ where each of the numbers c_i is either 0 or 1, and $c_n = 1$. The number $n = \deg(P)$ is the degree of the polynomial. All coefficients c_i for $i > n$ are considered to be zeroes.

The product of polynomials $P(x) = \sum_{i=0}^{n} p_i \cdot x^i$ and $Q(x) = \sum_{j=0}^{m} q_j \cdot x^j$ is the polynomial

$$S(x) = \sum_{i=0}^{n} \sum_{j=0}^{m} p_i \cdot q_j \cdot x^{i+j}.$$

Remember that in $\mathbb{F}_2$, the results of each addition and multiplication of the coefficients are immediately taken modulo 2.

The statement $P = 0$ is true if all coefficients of the polynomial $P(x)$ are zeroes.

You are given two polynomials P and Q over $\mathbb{F}_2$. Your task is to find two polynomials A and B such that the statement $AP + BQ \neq 0$ is true and the degree of $AP + BQ$ is minimal possible.

Input

The first line of input contains an integer T, the number of test cases ($1 \le T \le 100$). Each of the next T pairs of lines contains one test case. A test case consists of two polynomials. A polynomial is given as $n\ c_0\ c_1\ c_2\ \ldots\ c_n$. All c_i are equal to 0 or 1, and $c_n = 1$.

All given polynomials have positive degrees. The sum of all polynomial degrees in the input does not exceed 1000.

Output

For each test case, print two polynomials: first A, then B. The polynomials must be formatted as in the input. If there are several answers, print any one of them. The degrees of printed polynomials must be at most $\max(10, 2(\deg(P) + \deg(Q)))$. It is guaranteed that for any P and Q, there is an answer with the minimal degree of $AP + BQ$ satisfying the above constraint.

Example

polynoms.in	polynoms.out
3 2 1 1 1 1 1 1 4 1 0 0 0 1 2 1 0 1 4 1 0 1 0 1 3 1 1 0 1	0 1 1 0 1 0 0 0 1 2 0 1 1 3 1 0 1 1

Problem G. "Swap" Puzzle *(Division 2)*

Input file:	`puzzle-swap.in`
Output file:	`puzzle-swap.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

The "Swap" puzzle looks like a table of 4×4 cells which contain numbers from 1 to 16, each of them exactly once. Each number contains exactly two decimal digits: the numbers 1–9 have leading zeroes.

In one operation with the puzzle, one can take any two cells of the table and swap them.

Find an arbitrary shortest sequence of operations which transforms a given puzzle into the ordered table:

01	02	03	04
05	06	07	08
09	10	11	12
13	14	15	16

Input

The input consists of four lines corresponding to the rows of the table. Each line contains four numbers, and each of these numbers consists of exactly two digits. For that, the numbers 1–9 are given **with leading zeroes**. Consecutive numbers are separated by a space. It is guaranteed that each of the numberf from 1 to 16 occurs exactly once in the given table.

Output

On the first line, print one integer k: the number of operations. This number must be the minimal possible. On the next k lines, print the operations themselves.

To record the operations, we denote rows by letters `a`, `b`, `c`, `d` from top to bottom and columns by numbers `1`, `2`, `3`, `4` from left to right. Swapping two cells is recorded as r_1c_1-r_2c_2: row and column of one of the cells, the character «-» (minus, ASCII code 45), row and column of the other cell.

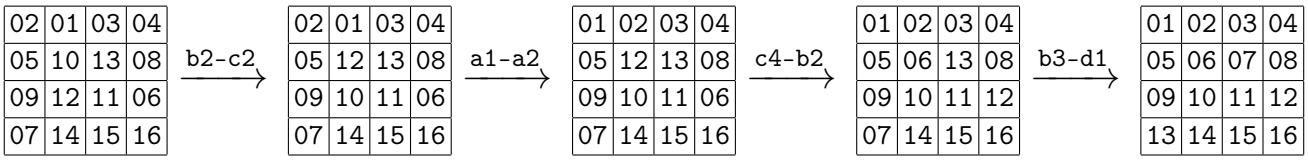
If there are several possible answers, print any one of them.

Example

puzzle-swap.in	puzzle-swap.out
02 01 03 04 05 10 13 08 09 12 11 06 07 14 15 16	4 b2-c2 a1-a2 c4-b2 b3-d1

Explanation

The sequence of operations in the example and the intermediate states of the table are shown below.



Problem H. Wrong Sieve *(Division 2)*

Input file:	`sieve.in`
Output file:	`sieve.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Many of you heard about Eratosthenes sieve. It is an algorithm which can find all prime numbers between 1 and N. The algorithm works in the following way.

Let us print all the numbers from 1 to N. On the first step, remove all numbers divisible by 2 except 2. On the second step, remove all numbers divisible by 3 except 3. On the k-th step, remove all numbers divisible by $k+1$ except $k+1$ (some of the numbers may have been removed earlier). It is not hard to show that after N steps, only prime numbers and 1 will still not be removed.

Let us consider one modification of this algorithm. On k-th step, remove every $(k+1)$-th number from the list of numbers which are **not yet removed**. So, on the first step, all even numbers will be removed. The numbers removed on the second step are $5, 11, 17, \ldots$. After infinitely many steps, the remaining sequence will start as $1, 3, 7, 13, 19, 27, 39, 49, \ldots$.

Your task is to check if the given number N is in the sequence, and if it is, find its position in the sequence, starting from 1.

Input

The first line of input contains an integer T: the number of test cases ($1 \le T \le 50$). The next T lines contain integers N_1, N_2, …, N_T for which you have to to solve the problem ($1 \le N_i \le 10^{12}$).

Output

For each test case, print one number — position of N_i in the sequence or -1 if it is not in the sequence.

Example

sieve.in	sieve.out
5 1 2 3 42 1359	1 -1 2 -1 42

Problem I. Space Cat *(Division 2)*

Input file:	`space-cat.in`
Output file:	`space-cat.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Cats are perfect space explorers because of their exceptional ability of landing on the feet. They are not afraid of changing gravity at all.

So, here is our famous solar Cat. He has a very important mission. The mission itself is to go through a space hypertunnel and bring *The Transcendence* to the people.

The space hypertunnel consists of two dimensions and has floor and ceiling. Both floor and ceiling consist of unit segments of different elevation. The Cat starts at the very first floor segment and must finish at the very last floor segment.

On any segment, the Cat may change the gravity and jump from the ceiling to the floor, or vice versa. This action needs as much energy as is the difference between ceiling and floor elevations on that segment.

Additionally, the Cat can move himself onto neighbouring segment, but only if it is possible to pass without climbing. Climbing could be dangerous in the tunnel! And, of course, our fluffy superhero cannot pass through walls.

You have to find the minimum amount of energy which is necessary to pass the hypertunnel.

Input

The first line contains the only integer n: the length of the hypertunnel ($1 \le n \le 10^5$). The second line consists of n space-separated integers: the ceiling levels c_i ($2 \le c_i \le 10^9$). The second line consists of n space-separated integers: the floor levels f_i ($1 \le f_i < c_i$).

Output

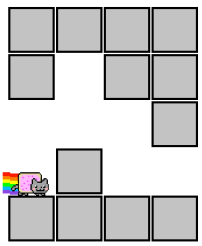
Print the only integer: the minimum amount of energy which is necessary to pass the hypertunnel, or -1 if it is impossible to do.

Examples

space-cat.in	space-cat.out
4 3 4 3 2 1 2 1 1	4
2 4 3 1 2	-1

Explanations

In the first example, the Cat must change the gravity, then go forward from segment 1 to segment 2, then turn the gravity back and go to the exit.



In the second example, there is no way to the exit.

Problem J. Tic-Tac-Toe Variation *(Division 2)*

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	256 mebibytes

This is an interactive problem.

Dima plays a variation of Tic-Tac-Toe game with Petya. A field consisting of 3×3 cells is used for the game. Initially, each cell is empty. The players take turns one after another, the first one to move is Dima. On each turn, the player has to put his symbol (`x` for Dima, `o` for Petya) in any empty cell.

Dima wins if he constructs a straight line of three `x`s: this may be a horizontal line, a vertical line, or one of the two diagonals. The rules for the second player however differ from the standard ones: Petya's goal is to **prevent Dima from constructing a straight line** of three `x`s. Formally, Petya wins if all cells of the field are already filled, but Dima did not yet win.

Your task is to play as Dima such that you always win. The judges' program will play as Petya.

Interaction Protocol

In each test, your program has to play from 1 to 100 games with the above rules as Dima. In different tests, the number of games and Petya's strategy may be different. Each game consists of several moves.

During a single game, the playing programs send the current state of the playing field to one another. This state is recorded as three lines each of which contains three characters. Character "`x`" (lowercase English letter ex) means that Dima's symbol is put in the respective cell, Character "`o`" (lowercase English letter oh) means Petya's symbol, and "`.`" (dot) signifies an empty cell.

When a game starts, the participant's program is given the initial state of the field: all cells are empty. To make a move, each playing program must print back the state it just recieved, changed according to the rules: exactly one of the empty cells must become a cell with the respective player's symbol.

A game is finished when one of the players wins. If the winner is Petya, evaluation stops with outcome "`Wrong Answer`". If Dima wins, the players start a new game if they have to play more.

In each test, the number of games and Petya's strategy are chosen in advance. If all planned games ended with Dima being the winner, instead of another empty board, the participant's program is given a board consisting **entirely of Dima's symbols** (`x`). When the participant's program gets such a board, it must terminate gracefully.

To prevent output buffering, flush the output buffer after each command you issue: this can be done by using, for example, `fflush (stdout)` in C or C++, `System.out.flush ()` in Java, `flush (output)` in Pascal or `sys.stdout.flush ()` in Python.

Example

standard input	standard output
... ...	
	... .x. ...
o.. .x. ...	
	ox. .x. ...
oxo .x. ...	
	oxo .x. .x.
... ...	
	... .x. ...
o.. .x. ...	
	o.. .x. .x.
oo. .x. .x.	
	oo. .x. xx.
ooo .x. xx.	
	ooo .x. xxx
xxx xxx xxx	

Explanation

In the example, which is also the first test in the testing system, the solution has to play two games. In each of them, Petya always chooses the uppermost empty cell and puts an o there. In case there are several such cells, Petya chooses the leftmost of them.

Please note that there are **no** empty lines in the actual input and output. The gaps are added so that all lines of input and output are displayed in the order in which they were transmitted.

Problem K. Captain Tarjan *(Division 2)*

Input file:	`treepaths.in`
Output file:	`treepaths.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

— *Nooo, here is another unsolvable tree problem...*
— *Then we need to call captain Tarjan for the rescue!*

Hypothetical dialogue during the contest

Not so long ago little Sergey started solving problems on trees. At first he learned about Aho-Hopcroft-Ullman-Tarjan algorithm that finds LCA in almost $O(1)$ time for offline queries, but this wasn't enough for him. "We need to go deeper," he thought. In one of the next Tarjan's articles, he found description of heavy-light decomposition which divides a tree into vertex-disjoint vertical paths. A path is called vertical if we can go along this path always travelling down (from the root to leaves). After that, we can calculate whatever we want on these paths.

"But what will happen if we make some another vertex the root of our tree? Then we can accidentally break verticality of some paths..." Sergey wondered. But Tarjan had an answer for this question too. Along with Sleator, he discovered Splay trees, which helped in building the link/cut tree data stucture. One of its key features is that we can modify paths is such a way that any vertex can become the root. This is exactly what we want! Maybe many years ago Tarjan was also as little as Sergey, and the same questions were in his head at that time?

But let us return to our problem. Obviously, Sergey started learning so much about trees not only because of pure curiosity. At his house, there is his favourite big tree with N vertices, and he wants to ask M very tricky queries of the form "calculate some function on the path in the tree that goes from u_i to v_i". How can we process such a query? Obviously, we need to divide our path into some smaller paths which all can be represented as subpaths of our vertex-disjoint vertical paths, and get all information we need from them. There is only one problem: not all edges can be located inside our decomposition, and therefore, we need to process them differently. All such edges connect two vertices from different paths. Sergey doesn't like corner cases, so he wants to minimize the total number of such edges in all of his queries.

Help Sergey to construct a valid decomposition that minimizes the number of such corner case edges. It is known that the root of his tree is always the vertex number 1.

Input

The first line contains two integers N and M ($1 \le N, M \le 10^5$). Next $N-1$ lines describe Sergey's tree: i-th of them contains two integers x_i and y_i which are the vertices of i-th edge ($1 \le x_i, y_i \le N$). The next M lines describe queries to some paths (u_i, v_i) in the same format.

Output

Output one number: the minimal number of edges outside of decomposition.

Example

treepaths.in	treepaths.out
4 3 1 2 2 3 2 4 1 3 1 4 3 4	2

Problem L. Gardening Lesson *(Division 2)*

Input file:	`unexpected-leaf.in`
Output file:	`unexpected-leaf.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Hexadecimal's new passion is gardening!

She found a tree which completely describes her fine spiritual organization. This tree consists of n vertices and, as many other ones, is a bidirectional graph with $n - 1$ edges and with no cycles.

The virus decided to grow a copy of the tree in her garden. And it's almost happened!

As everyone knows, it is necessary to check that all branches and leaves of the tree have grown successfully and no other things have grown. Unfortunately, one additional leaf has appeared unexpectedly.

Now you are asked to help in searching for the additional leaf to remove. Can you?

Input

The first line of input contains of the only integer n: the number of vertices in the original tree ($1 \le n \le 100$). The following $n - 1$ lines describe the original tree. The i-th of these lines contains two integers x_i and y_i: the numbers of vertices connected by an edge ($1 \le x_i, y_i \le n$). The following n lines describe the new tree which has $n+1$ vertices in the same format. It is guaranteed that the new tree can be produced from the original one by adding a leaf and renumbering the vertices.

Output

Print the only integer: the number of the additional leaf. If there are several possible solutions, find the one with the minimal number.

Examples

unexpected-leaf.in	unexpected-leaf.out
5 1 2 2 3 1 4 1 5 1 2 2 3 3 4 4 5 3 6	1
3 1 2 2 3 2 4 4 1 1 3	2
4 1 2 1 3 1 4 1 2 1 3 1 4 4 5	5