Problem A. Attractive Game

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	1024 megabytes

The legendary game developer known as FireToad works in a small indie company and is leading the development of the TOAD-2 project.

The versions of the project are numbered as follows: first comes the major version number, then a dot, then two digits of the minor version number, and then, if it exists, the number of the letter patch.

At the moment, there are major versions from 1 to 7. When the major version changes, the minor version is reset to zero.

At the moment, the minor version number has not exceeded the range from 00 to 89.

Then a letter may follow, indicating the letter patch within the minor version. So, if the game had version 7.34 before the letter patch, then after it, it has version 7.34b, and if, for example, version 7.34c, then after it, it has version 7.34d. Throughout the history of the game, within one minor version, there have not been more than seven letter patches.

You are given the number of an existing version of the game. Determine the possible numbers of the next versions (in case of a letter patch, a change in the minor version, and a change in the major version).

Input

The input contains the version number in the specified format. It is guaranteed that the major version number has a value from 1 to 7 inclusive, the minor version number is from 00 to 89 inclusive, and the letter of the letter patch (if any) is from '`b`' to '`h`' inclusive.

Output

Output three numbers of the next version — after a possible letter patch, after changing the minor version, and after changing the major version, respectively.

Examples

standard input	standard output
7.34d	7.34e 7.35 8.00
6.89	6.89b 6.90 7.00
3.14h	3.14i 3.15 4.00

Problem B. Back To Start

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	1024 mebibytes

There is a table with length n and width m.

A billiard ball begins to move from one corner with an angle of 45 degrees.

When will the ball bounce back to where it starts?

Formally, you are given n and m, and you need to calculate the return value of the following function.

```
int64_t check(int n, int m) {
  int x = 0, y = 0;
  int dx = 1, dy = 1;
  int64_t t = 0;
  while (1) {
    if (x + dx < 0) dx *= -1;
    if (x + dx > n) dx *= -1;
    if (y + dy < 0) dy *= -1;
    if (y + dy > m) dy *= -1;
    x += dx;
    y += dy;
    ++t;
    if (x == 0 && y == 0) break;
  }
  return t;
}
```

Input

The first line contains an integer t, the number of test cases ($1 \le t \le 10^5$). The test cases follow.

Each test case is described by a single line containing two integers n and m ($2 \le n, m \le 10^9$).

Output

For each test case, output a line containing one integer: the answer to the problem.

Example

standard input	*standard output*
5 2 2 2 3 2 4 2 5 2 6	4 12 8 20 12

Problem C. Chessboard And Two Pieces

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	1024 megabytes

You are playing a game with the computer on a standard 8×8 chessboard. Initially, the board is empty. The field designations on this board correspond to the standard chess notation: fields are denoted by a pair "letter-digit", for example, the bottom left field is denoted by `a1`.

```
8 . . . . . . . .
7 . . . . . . . .
6 . . . . . . . .
5 . . . . . . . .
4 . . . . . . . .
3 . . . . . . . .
2 . . . . . . . .
1 . . . . . . . .
  a b c d e f g h
```

The computer selects one of the white chess pieces (in this problem, a pawn is also considered a piece), which you do not know.

You choose any of the fields, except for the fields on the top and bottom horizontal, the computer places the piece on this field and provides a list of letters corresponding to the fields to which the piece can move. You can either immediately name the piece (if you guess it - you win), or ask to place another unknown piece on another field (the piece chosen the second time may coincide with the original one or differ from it; in particular, it is possible that both pieces will be queens or kings) and provide a list of letters corresponding to the fields to which the first and second pieces can move after the second piece has been placed. If you requested the second piece, you must guess **both** pieces.

Please note that the white pawn can only move 1 position forward (i.e. towards increasing the number of the horizontal, remaining on the same vertical), the rook — any number of positions horizontally or vertically, the bishop - any number of positions diagonally, the knight - in the shape of a letter L: 2 squares in one direction, then — 1 in the perpendicular, the king — to any of the 8 fields, sharing a common point with the current one, the queen — any number of positions horizontally or diagonally.

Interaction Protocol

The interaction starts with your program, outputting a question mark `?`, and after that, through a space, the coordinates of the field in standard chess notation — a string in which the first character is a letter from '`a`' to '`h`', and the second is a number from 2 to 7 (using the first and eighth horizontal is prohibited to exclude putting the pawns on them).

The computer outputs a string consisting of no more than eight letters from '`a`' to '`h`', sorted in ascending order - the numbers of the verticals to which the piece can move. If the piece cannot make a move, a single character '`-`' is output.

After that, your program either outputs the answer in the form of `!` f_1, where f_1 is `p` for pawn, `n` for knight, `b` for bishop, `r` for rook, `k` for king and `q` for queen, or makes a second request in a similar format with the distinct field.

On the second request, the computer outputs two lines — the numbers of the verticals to which the first piece can move (taking into account possible interference from the second), and the numbers of the verticals to which the second piece can move (taking into account possible interference from the first).

After that, your program must output the answer in the form of `!` f_1 f_2, where f_1 and f_2 are the designations for the first and second pieces, respectively.

It is guaranteed that the computer does not replace the already placed piece on the board with another, even if the answers for them are the same (i.e. the interactor is not adaptive).

Example

standard input	standard output
? e5 ? b2 ! k q	 def def abcdefgh

Problem D. Draw The Curve

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	1024 mebibytes

There are n base points on the plane, with some segments connecting them.

It is guaranteed that every two base points do not coincide, every three base points are not collinear, and segments only intersect at endpoints.

There are also m source points on the plane.

It is guaranteed that every source point is different from each of the n base points and does not lie on any segment.

The question is, for each segment, whether you can draw a curve from some source point to the midpoint of this segment without intersecting any other segments.

Input

The first line contains three integers n, m, e ($1 \le n, m \le 100$; $0 \le e \le 300$), denoting the number of base points, source points, and segments.

Each line of the next n lines contains two integers x, y ($|x|, |y| \le 10^9$), denoting a base point (x, y).

Each line of the next m lines contains two integers x, y ($|x|, |y| \le 10^9$), denoting a source point (x, y).

Each line of the next e lines contains two integers i, j, denoting a segment connecting the i-th base point and the j-th base point ($1 \le i < j \le n$).

Output

Print a string of length e. The i-th character of the string must be "1" if you can draw a curve from some source point to the midpoint of the i-th segment without intersecting any other segments, or "0" otherwise.

Example

standard input	*standard output*
4 1 3 -2 0 0 2 2 0 0 1 0 3 1 2 2 3 1 3	111

Problem E. Effective Pricing

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	1024 mebibytes

The sharp rise in inflation in Byteland led to a decrease in demand for diamonds. So it's no wonder that a new marketing director arrived at the Bytelandian diamond supermarket.

His first decision was to change the pricing policy to a more attractive one. Before that, all prices in the supermarket were expressed as whole numbers of burlas. The new director decided to adopt proven advanced techniques and proposed to reduce each price by 1 bitcent (1 bitcent = 1/100 bytalers).

The price tags in the supermarket are made up of plastic digits, so to implement the director's order, it is necessary to purchase a certain number of "nines". Based on the current price list of the supermarket's products, determine how many additional "nines" will be needed to record the prices after the "unprecedented reduction".

Input

The first line of the input contains a single integer N ($1 \le N \le 1000$) — the number of positions in the supermarket. Each of the following lines contains a single integer p_i ($1 \le p_i < 10^{1000}$) — the original price of the next position in bytalers.

Output

Output a single integer: the number of additional nines in the price list after the marketing director's proposal is implemented.

Example

standard input	*standard output*
3 15 10 2023	7

Problem F. Find The Ratio

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	1024 megabytes

Alice is interested in competitive programming. She participates in various competitions, which allows her to adequately assess her chances in the competitions by looking at the composition of the participants. So, before the qualification for the PopCoder Open competition, Alice found out that her chances of reaching the final are equal to A percent, where A is a real number from 1 to 100, the decimal representation of which contains no more than two digits after the decimal point.

Alice wants to represent the chances as a rational number p/q, where p and q are coprime positive integers, the decimal fraction p/q when all decimal places after the third are discarded, coincides with A, and the value of q is as small as possible.

Help her find this representation.

Input

The first line of the input contains a single integer T ($1 \le T \le 200$) — the number of test cases. Each test case is given in a separate line and contains a real number A ($1 \le A \le 100$, $100 \cdot A$ is an integer) — Alice's chances of reaching the final in percentage.

Output

For each test case, output two integers p and q in a separate line — the numerator and the denominator of the required representation.

Example

standard input	standard output
3 42 20.5 3.98	42 1 41 2 199 50

Problem G. Generate Interesting Numbers

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	1024 mebibytes

The mixed-base positional number system is defined by a sequence of bases b_i, where b_i is an integer, not less than 2.

To obtain the representation of the number X in such a number system, the following algorithm is applied: $x_0 = x$, $a_i = x_i \bmod b_{i+1}$, $x_{i+1} = \lfloor x_i / b_{i+1} \rfloor$.

In other words, numbers in such a system are represented as

$$a_0 + b_1 \cdot (a_1 + b_2 \cdot (a_2 + \ldots + b_{n-1} \cdot a_{n-1}) \ldots))$$

where $0 \le a_i < b_{i+1}$ for all i from 0 to $n - 1$.

Thus, the maximum number that can be represented in the system defined by the sequence of b_i is equal to $\prod_{i=1}^{n} b_i - 1$.

We will call a positive integer k *interesting* if the following rule is true: a number in the number system defined by the sequence b_i is divisible by k if and only if the sum of the digits of this number (i.e., the sum of all a_i) is also divisible by k (analogous to divisibility by 3 and 9 in the decimal number system). In particular, since all numbers are divisible by 1, 1 will be a interesting number for any sequence of bases.

Given the sequence of bases b_i, find all interesting numbers for the corresponding mixed number system.

Input

The first line of the input contains a single integer N - the length of the sequence of bases ($2 \le N \le 10^5$).

The second line contains N integers b_i ($2 \le b_i \le 10^9$) - the elements of the sequence of bases, listed in the same order as they appear in the sequence.

Output

Output all interesting numbers for the given sequence, sorted in ascending order, one number per line.

Examples

standard input	*standard output*
3 10 10 10	1 3 9
4 15 10 20 23	1

Problem H. Highest Game Score

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	1024 mebibytes

Alice and Bob like playing games. Today they play on a special grid with 2 rows and n columns. Alice and Bob each have a chess piece, starting at $(1, 1)$ and $(2, n)$, respectively. They will take turns moving their chess piece, with Alice going first.

In each turn, they can choose to stay still or move the chess piece to any point adjacent horizontally or vertically which has not been visited by the other's piece. The game will end after $10^{10^{10}}$ turns.

Every point in this grid has a non-negative weight. For each player, the score that he/she gets is the sum of the weights of all the points his/her chess piece has visited. The weight is counted only once, even if the piece visited a point multiple times.

Both Alice and Bob want to maximize the score that they get. As a spectator, you want to know the score that Alice gets if both of them play optimally.

Input

The first line contains an integer t, the number of test cases ($1 \le t \le 5 \cdot 10^4$). The test cases follow.

The first line of each test case contains a single integer n representing the size of the grid ($1 \le n \le 10^5$).

The second line of each test case contains n integers $a_{1,1}, a_{1,2}, \ldots, a_{1,n}$. The i-th of them represents the weight of point $(1, i)$.

The third line of each test case contains n integers $a_{2,1}, a_{2,2}, \ldots, a_{2,n}$. The i-th of them represents the weight of point $(2, i)$.

It is guaranteed that $0 \le a_{i,j} \le 10^9$, and the sum of n across all test cases will not exceed $2.5 \cdot 10^5$.

Output

For each test case, print a line with a single integer: the score that Alice gets if both players play optimally.

Example

standard input	*standard output*
4 2 1 4 2 1 3 1 1 4 5 1 4 4 1 9 4 9 1 0 0 1 7 3 1 4 1 5 9 2 6 5 3 5 8 9 8	5 6 15 25

Problem I. Interactions With Graph

Input file:	*standard input*
Output file:	*standard output*
Time limit:	15 seconds
Memory limit:	1024 mebibytes

This is an interactive problem. You have to use the `flush` *operation right after printing each line. For example, you can use the function* `fflush(stdout)` *for C or C++,* `System.out.flush()` *for Java,* `flush(output)` *for Pascal, and* `sys.stdout.flush()` *for Python.*

You are an ordinary employee of a data analysis department. However, it seems that your colleagues are geniuses at making trouble. Just now they made some trouble again.

They downloaded a large graph from the remote server and wrote a program to analyze the graph. They nearly finished the analysis and thought the graph should be no longer useful. So they let their program add some new edges to the graph without any backups. But then they changed their mind and want to compute the longest path in the original graph. Downloading the graph again costs too much time. Now it is your time to save the day.

The original graph on the remote server is a directed acyclic graph $G = (V, E)$. All edges are unit length. The input of your program is the modified version $G' = (V, E')$. It is guaranteed that $E \subseteq E'$, **and G' is a directed acyclic graph**.

Your program should output $\ell(G)$, the length of the longest path in G. The length of a path equals the number of edges in the path.

To test whether an edge belongs to the original graph, your program can make queries to the remote server. To query the existence of edge $u \to v$, output "`? u v`". Flush the output stream after printing each query. The remote server will respond `1` if edge $u \to v$ belongs to the original graph, and `0` otherwise.

After your program gets the answer, print "`! ans`", where $ans = \ell(G)$, and **terminate your program normally** immediately after flushing the output stream.

Your program is allowed to make no more than $(|E'| - |E| + 1) \cdot (\ell(G) + 1)$ queries (not including printing the answer) to the remote server, although $|E|$ and $\ell(G)$ are unknown to you.

Input

Use standard input to read the responses to the queries.

The first line contains two integers n and m ($1 \le n \le 5 \cdot 10^4$; $1 \le m \le 10^5$): the number of vertices and edges of graph G'.

Each of the next m lines contains two integers u and v ($1 \le u, v \le n$) specifying a directed edge $u \to v$ in graph G'. No edge appears more than once.

It is guaranteed that $0 \le |E'| - |E| \le 200$.

The following lines will contain responses to your queries. Each response is either "`0`" or "`1`". The i-th of these lines is a response to your i-th query.

After answering $(|E'| - |E| + 1) \cdot (\ell(G) + 1)$ queries, the remote server no longer responds.

The testing system will allow you to read the response to a query only after your program prints the query and performs the `flush` operation.

Output

To make the queries, your program must use standard output.

Your program must print the queries in the form of "`? u v`", one query per line. Do not forget to end the line after each query. Your program must guarantee that $1 \le u, v \le n$. After printing each line your program must perform the `flush` operation.

The response to the query will be given in the standard input after you flush the output. In case your program finds the answer, print a line "`!` *ans*", where $ans = \ell(G)$, and terminate your program.

Example

standard input	*standard output*
5 5 1 2 1 3 2 5 3 4 4 5 1 1 1 0 1	? 1 2 ? 1 3 ? 2 5 ? 3 4 ? 4 5 ! 2

Problem J. Joy With Spell

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 second
Memory limit:	1024 mebibytes

Alice loves playing HearthStone! She loves the hero class of Warlock, who can cast the spell named Defile.

When cast, Defile deals 1 unit of damage to the health of all minions. If any minion dies, Defile will be cast again automatically. Importantly, if two or more minions die simultaneously, it still causes a single Defile cast. That, in turn, may kill other minions, causing Defile to be cast again, and so on.

The health of each minion is a nonnegative integer. A minion dies when their health becomes zero. If a minion dies, it will disappear. It will not die twice.

Now there are n minions. Before casting Defile, Alice can make zero or more steps. In each step, Alice changes a single minion's health by one. That is to say, if the health of a minion is x, Alice can change it to $x - 1$ or $x + 1$.

Alice wants to know the minimum number of steps such that, after these steps, she can cast a single Defile to kill all the minions.

Input

The first line contains a single integer n ($1 \le n \le 10^6$).

The next line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^6$), the health of the n minions.

Output

Print one integer: the minimum number of steps before Alice can cast a single Defile to kill all the minions.

Example

standard input	*standard output*
6 4 6 8 9 2 4	12

Problem K. King Of Byteotia

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	1024 mebibytes

You are the king of the Byteotian Kingdom. The Byteotian Kingdom consists of n cities and $n-1$ two-way roads, each connecting a pair of cities. It is guaranteed that it is possible to traverse between any two cities through the roads.

Each road has a color, black or white, and a length ℓ_i. Initially, each road is white. However, you think that it is too boring this way. So you have decided to assign m robots to m cities to paint the roads to your favorite color pattern. Different robots **can** be assigned to the same city. Robot i starts at city p_i, travels through some roads (possibly none), and then stops. A robot **cannot** travel through one road multiple times. When a robot travels through a road, it flips the color of the road (if it was white, it turns black, and vice versa). The robots are independent, which means that they will not interfere with each other in the travel. We assume that different robots don't paint any road simultaneously. Also, different robots **can** stop in the same city. The cost of a robot's travel is defined as the sum of lengths of all the roads on its path.

As the king of the Byteotia, you want to minimize the total cost of all the travels. If it is impossible to paint the roads to the desired pattern with the m robots, print -1 instead.

Input

The first line contains an integer t, the number of test cases ($1 \le t \le 5000$). The test cases follow.

The first line of each test case contains two integers n and m ($2 \le n \le 5000$ and $1 \le m \le 5000$), denoting the number of cities and the number of robots, respectively.

Each of the next $n-1$ lines contains four integers u_i, v_i, ℓ_i, c_i ($1 \le u_i < v_i \le n$; $1 \le \ell_i \le 10$; $c_i = 0$ or $c_i = 1$), denoting a road of length ℓ_i connecting cities u_i and v_i. If $c_i = 0$, you should paint it white in the desired pattern; otherwise, you should paint it black. It is guaranteed that it is possible to traverse between any pair of cities through the given roads.

Then a single line contains m integers p_j ($1 \le p_j \le n$), denoting the starting city for each robot.

It is guaranteed that the of sum of n over all test cases will not exceed 5000, and the sum of m over all test cases will not exceed 5000.

Output

For each test case, print one line containing a single integer: the minimal total cost to paint all the roads to the desired pattern. If it is impossible to do so, print -1 instead.

Example

standard input	*standard output*
5 3 2 1 2 1 1 2 3 2 1 1 3 4 2 1 2 3 1 2 3 1 0 3 4 4 1 1 2 5 4 1 2 3 0 2 3 1 1 3 4 2 0 4 5 2 1 1 1 1 1 5 2 1 2 2 1 1 3 3 0 1 5 2 1 3 4 1 1 1 2 10 5 1 2 10 1 2 3 3 1 3 4 4 0 4 5 4 1 5 6 2 1 2 7 8 0 2 8 9 1 4 9 1 0 1 10 4 0 10 10 2 1 8	3 9 21 -1 42