Problem A. Absolutely Flat

Time limit: 2 seconds
Memory limit: 512 megabytes

Alice is a proud owner of a four-legged table, and she wants her table to be flat. Alice considers the table to be flat if its four legs have equal lengths.

Alice measured the table's current leg lengths and got a_1, a_2, a_3, and a_4. She also has a pad of length b. Alice can attach the pad to one of the legs, in which case the length of that leg will increase by b. She can also decide not to attach the pad, in which case the lengths of the legs will not change. Note that Alice has just a single pad, so she can neither apply it twice to the same leg nor apply it to two different legs.

Find out whether Alice can make her table flat.

Input

The input contains five positive integers a_1, a_2, a_3, a_4, and b, each on a separate line — the lengths of the table's legs, and the length of the pad Alice has ($1 \le a_i, b \le 100$).

Output

Print 1 if Alice can make her table flat, and 0 otherwise.

Examples

standard input	standard output
10 10 10 10 5	1
13 13 5 13 8	1
50 42 42 50 8	0
20 40 10 30 2	0

Note

In the first example test, the table is already flat, no pad is needed.

In the second example test, Alice can apply the pad to the third leg to make the table flat.

In the third and the fourth example tests, Alice can not make her table flat.

Problem B. Bricks in the Wall

Time limit: 2 seconds
Memory limit: 512 megabytes

Bob is decorating a loft-style rectangular wall with bricks. The wall consists of $n \times m$ unit cells. Some cells are already occupied by bricks, while the remaining cells are empty.

Bob wants to add up to two more bricks to this wall. New bricks must have a width equal to 1 unit and can have any positive integer length. Each brick can only be placed horizontally or vertically, so each new brick will occupy several consecutive empty cells in one row or in one column. Also, these two bricks must not intersect, i.e. occupy the same cell.

What is the maximum possible sum of lengths of at most two new bricks that Bob can add to this wall?

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). The description of the test cases follows.

The first line of each test case contains two integers n and m — the height and the width of the wall ($1 \le n, m$; $n \cdot m \le 10^6$).

The next n lines contain m characters each, describing the wall. An occupied cell is denoted by '`#`', an empty cell is denoted by '`.`'.

It is guaranteed that the sum of $n \cdot m$ over all test cases does not exceed 10^6.

Output

For each test case, print a single integer — the maximum possible sum of lengths of at most two new bricks.

Example

standard input	standard output
5 2 2 .. .. 4 5 ###.# #.... .##.# #.#.# 2 1 . . 2 3 ### #.# 5 4 ##.# ..#. #.#. .... #.##	4 6 2 1 7

Problem C. Computer Network

Time limit: 2 seconds
Memory limit: 512 megabytes

Cupa is building a connected network using n computers and a single hub.

The computers are numbered from 1 to n. Each computer i has an outgoing wire that can transfer one bit of data to the other end in d_i milliseconds.

The hub has k ports into which the computer's wires can be connected, and each computer has a single port.

Cupa requires each computer's wire to be connected to some port — either in the hub or in another computer. It should also be possible to send data to the hub from every computer, either directly or via other computers.

The network latency t_i for each computer i is defined as the time it takes to send one bit of data from computer i to the hub. We will assume that it takes no time for intermediate computers to redirect received data to their own outgoing wires.

After the network is built, Cupa will calculate the network latency t_i for each computer i. He wants the total network latency over all computers, i.e. $t_1 + t_2 + \ldots + t_n$, to be as small as possible.

Help Cupa to build the network in a way that minimizes the total network latency.

Input

The first line contains two integers n and k — the number of computers and the number of ports in the hub ($1 \le k \le n \le 100$).

The second line contains n integers $d_1, d_2, \ldots, d_n$ — the list of data transfer times through each computer's wire ($1 \le d_i \le 100$).

Output

Print a single integer — the minimum possible total network latency.

Examples

standard input	standard output
3 2 20 30 10	70
5 1 10 10 10 10 10	150
5 2 10 10 10 10 10	90
6 3 5 6 2 3 1 4	27

Note

In the first example test, Cupa should connect computers 2 and 3 to the hub, and connect computer 1 to computer 3. In this case, $t_1 = 20 + 10 = 30$, $t_2 = 30$, and $t_3 = 10$. The answer is $t_1 + t_2 + t_3 = 70$.

In the second example test, the computers should be connected in a chain leading to the hub in arbitrary order. The total network latency is $10 + 20 + 30 + 40 + 50 = 150$.

Problem D. Dice Grid

Time limit: 2 seconds
Memory limit: 512 megabytes

Debora is playing a video game. In one of the levels, she is given a cube and a flat colorful $n \times n$ grid. The cell in row i and column j of the grid is denoted by (i, j) and has color $c_{i,j}$. Debora can see the whole grid, including the color of each cell.

The cube face size matches the grid cell size. Whenever we say that the cube is located at cell (i, j), it means that its *bottom* face coincides with the grid cell (i, j). Opposite to the bottom face is the *top* face. The face that is "looking" at cell $(i + 1, j)$ is called the *front* face. The *back* face is "looking" at cell $(i - 1, j)$, the *right* face is "looking" at cell $(i, j + 1)$, and the *left* face is "looking" at cell $(i, j - 1)$.

Initially, the cube is located at cell $(1, 1)$. The goal of the game is to roll the cube to cell (n, n).

From any cell (i, j), Debora can only move the cube down — to cell $(i + 1, j)$, or right — to cell $(i, j + 1)$. The way to move the cube down is to rotate it around the edge connecting its bottom and front faces. For instance, after the rotation, the front face becomes the new bottom face. Similarly, the way to move the cube to the right is to rotate it around the edge connecting its bottom and right faces.

The faces of the cube are not colored yet. Debora has to paint each face in any color she wants. At every moment of the game, including the moments when the cube is located at $(1, 1)$ and (n, n), the cube's bottom face color has to match the color of the grid cell where the cube is located.

The goal is to paint the cube in such a way that Debora will be able to move the cube from cell $(1, 1)$ to cell (n, n) satisfying the conditions above. Find any possible cube coloring.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 625$). The description of the test cases follows.

The first line of each test case contains a single integer n — the number of rows and columns in the grid ($2 \le n \le 50$).

The i-th of the following n lines contains n integers $c_{i,1}, c_{i,2}, \ldots, c_{i,n}$ ($0 \le c_{i,j} < 2^{24}$). The RGB color of cell (i, j) is $c_{i,j}$.

It is guaranteed that the sum of n^2 over all test cases does not exceed 2500.

Output

For each test case, if no coloring exists, print a single word "`No`" on a separate line.

Otherwise, in the first line, print a single word "`Yes`".

In the second line, print six integers a_b, a_l, a_k, a_f, a_r, and a_t — the colors of the bottom, left, back, front, right, and top faces of the cube, respectively, in its initial position at cell $(1, 1)$ ($0 \le a_i < 2^{24}$).

If several possible colorings exist, print any of them.

Example

standard input	standard output
4 2 1 1 0 0 3 1 2 3 9 6 4 7 8 1 4 1 2 3 4 9 8 7 5 10 8 12 2 13 14 15 6 4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16	Yes 1 3 4 0 0 5 Yes 1 10 10 4 2 3 Yes 1 4 6 5 2 3 No

Note

In the third example test case, the cube can be moved from $(1, 1)$ to $(4, 4)$ with the following sequence of moves: right, right, right, down, down, and down.

Problem E. Easily Distinguishable Triangles

Time limit: 2 seconds
Memory limit: 512 megabytes

Eva loves painting. Today she is working with a square canvas of $n \times n$ unit cells. Each cell is painted white, painted black, or empty — not painted at all.

Eva is going to draw a black triangle inside each empty cell. She wants each triangle to be right-angled and have an area of $\frac{1}{2}$ square unit cells. Thus, there are four ways to draw a single triangle:

Each triangle is a piece of art, and Eva wants them to be easily distinguishable from the rest of the painting. To achieve that, no two black triangles may share a common side with each other, and no black triangle may share a common side with a black square. Note that two black squares are allowed to share a common side.

Help Eva to find out how many ways there are to finish her painting. Since the number can be large, calculate it modulo 998 244 353.

Input

The first line contains a single integer n — the side length of the canvas ($1 \le n \le 1000$).

The next n lines describe the canvas from top to bottom. The i-th of these lines contains n characters $s_{i,1}, s_{i,2}, \ldots, s_{i,n}$. If $s_{i,j} =$ '`.`', the cell in the i-th row and the j-th column of the canvas is painted white. If $s_{i,j} =$ '`#`', that cell is painted black. If $s_{i,j} =$ '`?`', that cell is empty.

Output

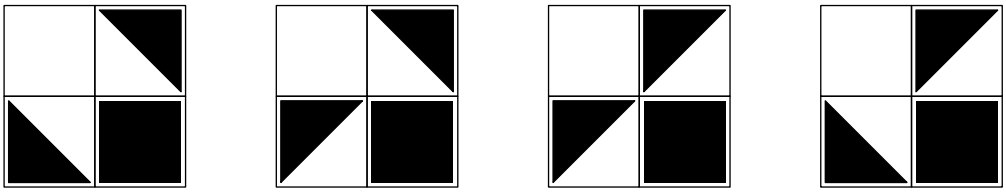
Print a single integer denoting the number of ways to finish Eva's painting, modulo 998 244 353.

Examples

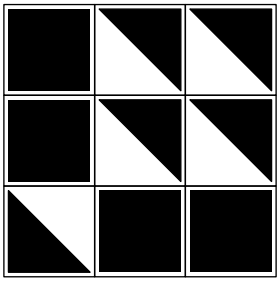
standard input	standard output
2 .? ?#	4
3 #?? #?? ?##	1
3 .#. #?# .#.	0

Note

In the first example test, there are 4 ways to finish the painting, as illustrated below:



In the second example test, there is a single way to finish the painting:



In the third example test, regardless of how Eva draws the triangle in the center cell, it will share two sides with black squares.

Problem F. Focusing on Costs

Time limit: 2 seconds
Memory limit: 512 megabytes

In modern gadgets, it's crucial to trim down the fat and get rid of unnecessary features, like a headphone jack. The same trend applies to the calculator industry.

In their pursuit for minimalism, Cosio calculator company started to produce calculators that have a single display and can only compute trigonometric functions `sin`, `cos`, `tan` and their inverses `asin`, `acos`, `atan`.

Initially, the calculator's display shows the number 0. After that, for each of the functions listed above, you can press a button that applies that function to the displayed number. If the operation is inapplicable or produces infinity, then the calculator breaks and stops responding.

You took it as a challenge to figure out what you can achieve using this calculator. Find a way to compute $\frac{a}{b}$ using at most 1000 operations.

Input

The only line contains two integers a and b ($1 \le a, b \le 10$).

Output

In the first line, print a single integer k — the number of button presses in your solution ($1 \le k \le 1000$).

In the second line, print the applied operations in order, separated by spaces.

The solution will be checked with a program in C++ using the standard 64-bit floating-point type: `double`. Your answer will be considered correct if the sequence of actions does not cause an error, and in the end the calculator displays $\frac{a}{b}$ with an absolute error of at most 10^{-9}.

You do not have to find the shortest solution. Any solution satisfying the constraints will be accepted.

Examples

standard input	standard output
1 1	4 atan cos sin asin
2 1	11 cos atan sin atan sin atan sin atan sin acos tan

Problem G. Greatest Common Divisor

Time limit: 5 seconds
Memory limit: 512 megabytes

Gennady is an aspiring programmer. He is currently learning the Euclidean algorithm for computing the greatest common divisor of two positive integers.

Unfortunately, Gennady sometimes confuses the integer division operator (denoted by `div`) with the remainder operator (denoted by `mod`). As an example, 37 `div` 10 = 3 and 37 `mod` 10 = 7.

Here's Gennady's latest implementation of the Euclidean algorithm:

- *Input: two positive integers x and y.*
- *While $y > 0$:*

  Set $x = x$ `div` y, then swap x and y.
- *Output: x.*

As you can see, if Gennady used the `mod` operator instead of the `div` operator, his implementation would be correct: the algorithm above would successfully find the greatest common divisor of x and y. However, it turns out that even with this nasty bug the algorithm sometimes works correctly!

You are given an integer n. Gennady is interested in finding all input pairs (x, y) such that $1 \le x, y \le n$, the algorithm finishes, and produces the correct output. Let $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$ be all such pairs in lexicographic order (for all $1 \le i < k$, either $x_i < x_{i+1}$, or $x_i = x_{i+1}$ and $y_i < y_{i+1}$).

You are also given q queries. Query i is a positive integer p_i, and you should print x_{p_i} and y_{p_i}, or report that $p_i > k$.

Input

The first line contains two integers n and q — the upper bound on the input values and the number of queries ($1 \le n, q \le 2 \cdot 10^5$).

Each of the next q lines contains a single integer p_i ($1 \le p_i \le n^2$).

Output

For each query, print two integers. These integers must either be x_{p_i} and y_{p_i}, denoting the p_i-th input pair in lexicographic order such that the algorithm finishes and produces a correct output, or `-1 -1` if there are less than p_i such pairs.

Example

standard input	standard output
10 13 1 2 3 4 5 6 7 8 9 10 11 12 13	2 2 3 3 4 2 4 4 5 5 6 6 7 7 8 8 9 3 9 9 10 4 10 10 -1 -1

Problem H. Hidden Digits

Time limit: 5 seconds
Memory limit: 512 megabytes

You are given a sequence of n digits d_0, d_1, $\ldots d_{n-1}$. Find the minimum positive integer x such that for all $0 \le i < n$, the decimal representation of number $x + i$ contains the digit d_i.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^5$). The description of the test cases follows.

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

The second line contains a string of n digits $d_0 d_1 \ldots d_{n-1}$ ($0 \le d_i \le 9$).

It is guaranteed that the sum of n over all test cases does not exceed 10^6.

Output

For each test case, print a single integer x — the smallest positive integer such that the decimal representation of $x + i$ contains the digit d_i for all $0 \le i < n$.

Example

standard input	standard output
6 5 12345 5 01234 3 239 9 998244353 10 1000000007 20 18446744073709551616	1 10 92 45296 701 10367486

Problem I. IQ Game

Time limit:	2 seconds
Memory limit:	512 megabytes

A popular TV show "Kak? Zachem? Pochemu?" highlights a team of six players working together to solve challenging questions. Players sit around a circular table divided into n sectors numbered clockwise from 1 to n. At the start of the game, each sector contains an envelope with a question to be answered.

Each round, the spinning top at the center of the table chooses a sector of the table uniformly at random. If the chosen sector contains an envelope, the host opens it and reads the question inside. If there is no envelope in the chosen sector, the host opens the next envelope in the clockwise direction from the chosen sector instead. After the round, the opened envelope is removed from the table.

Tonight, the audience's favorite team is playing. They have already played $n - k$ rounds out of n, so there are k envelopes remaining on the table. Things are not looking good for the team — one more incorrect answer will send them home. One of the questions is a special, notoriously hard question called "Hyperblitz". The team is confident they can answer each of the remaining questions except "Hyperblitz". Find the expected number of rounds they will play, modulo 998 244 353 (see the Output section for details).

Input

The first line contains three integers n, k, and s — the total number of sectors, the number of remaining questions, and the sector containing the "Hyperblitz" question ($1 \le n \le 10^9$; $1 \le k \le \min(n, 200)$; $1 \le s \le n$).

The second line contains k distinct integers $q_1, q_2, \ldots, q_k$ — the numbers of sectors that still have envelopes, in clockwise order ($1 \le q_1 < q_2 < \ldots < q_k \le n$).

There is exactly one index i with $q_i = s$.

Output

Print a single integer — the expected number of rounds the team will play (including the inevitable "Hyperblitz"), modulo 998 244 353.

Formally, let $M = 998\,244\,353$. It can be shown that the expected number of rounds can be expressed as an irreducible fraction $\frac{p}{q}$, where p and q are integers and $q \not\equiv 0 \pmod{M}$. Print the integer equal to $p \cdot q^{-1} \bmod M$. In other words, print such an integer x that $0 \le x < M$ and $x \cdot q \equiv p \pmod{M}$.

Examples

standard input	standard output
3 2 3 2 3	665496237
6 3 4 1 2 4	582309208
8 8 5 1 2 3 4 5 6 7 8	499122181

Note

In the first example test, in the first round, the team plays the "Hyperblitz" with probability $\frac{1}{3}$, so with probability $\frac{1}{3}$ they play 1 round, and with probability $\frac{2}{3}$ they play 2 rounds. The expected number of rounds is $1 \cdot \frac{1}{3} + 2 \cdot \frac{2}{3} = \frac{5}{3}$.

As $3^{-1} \bmod 998\,244\,353 = 332\,748\,118$, the correct output is $5 \cdot 332\,748\,118 \bmod 998\,244\,353 = 665\,496\,237$.

Problem J. Joking?

Time limit:	2 seconds
Memory limit:	512 megabytes

Julia wants to create a new board game for n players. As part of the game, players decide the order of their turns. The game should be fair: every possible permutation of players should be chosen with the same probability.

To help players determine this permutation, Julia wants to create n different k-sided dice. Each player will throw their own dice and look at the number. The player with the smallest number will go first, the player with the second smallest number will go second, and so on. To make sure no ties could happen, all numbers used on all dice should be distinct.

That could be a good math problem, but this is a programming contest, so we allow some imprecision. We ask you to create the dice for this game, but the probabilities of obtaining the permutations may differ slightly. Your solution will be accepted if the relative difference of probabilities of any two permutations is no more than 0.2%.

Formally, there are k^n different outcomes of throwing all n dice. For each permutation P, we can compute the number of scenarios $f(P)$ that lead to this permutation. For any two permutations P and Q, the following should be true: $\frac{|f(P)-f(Q)|}{max(f(P),f(Q))} \le 0.002$.

You may choose any k, but it may not exceed 120.

Input

The only line contains a single integer n — the number of players ($2 \le n \le 5$).

Output

In the first line, print a single integer k — the number of sides on each dice ($1 \le k \le 120$).

Each of the next n lines should describe one dice. For each dice, print k integers from 1 to $k \cdot n$. All integers used on all dice should be distinct.

Examples

standard input	standard output
2	2 1 4 2 3
3	16 3 7 9 10 12 17 18 19 28 32 33 35 38 40 43 48 1 2 6 13 14 20 22 26 27 29 30 36 37 39 44 46 4 5 8 11 15 16 21 23 24 25 31 34 41 42 45 47

Note

In the first example test, both permutations of players have the probability of $\frac{1}{2}$.

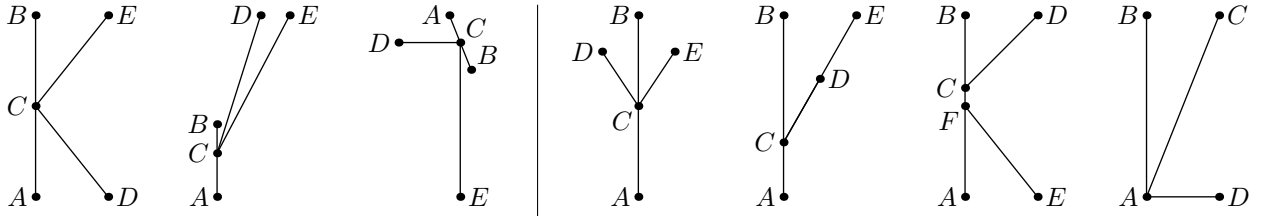
In the second example test, there are $16^3 = 4096$ possible scenarios. Permutations $[2, 1, 3]$ and $[3, 1, 2]$ arise in 682 scenarios each, while every other permutation arises in 683 scenarios. Thus, the relative difference between the most and the least probable permutations is $\frac{683-682}{683} \approx 0.146\%$.

Problem K. K-Shaped Figures

Time limit: 3 seconds
Memory limit: 512 megabytes

Let's say that three segments on a plane form a *K-shaped figure* if:

- two of them share a common endpoint;
- this common endpoint lies strictly inside the third segment;
- these two segments are located on the same side with respect to the third one;
- all three segments are pairwise not collinear.



You are given a collection of n segments on the plane. Find the number of triples of segments from this collection that form a K-shaped figure.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 3333$). The description of the test cases follows.

The first line of each test case contains a single integer n — the number of segments ($3 \le n \le 1000$).

The i-th of the following n lines contains four integers $x_{i,1}$, $y_{i,1}$, $x_{i,2}$, $y_{i,2}$ — the coordinates of endpoints of the i-th segment ($-10^6 \le x_{i,1}, y_{i,1}, x_{i,2}, y_{i,2} \le 10^6$). All segments have positive lengths. Some segments may coincide.

It is guaranteed that the sum of n over all test cases does not exceed 10^4.

Output

For each test case, print a single integer — the number of triples of segments that form a K-shaped figure.

Example

standard input	standard output
2 5 0 0 0 10 0 5 3 10 0 5 3 0 0 5 7 4 0 5 6 2 8 0 0 10 10 3 4 4 4 4 4 4 5 3 4 4 4 7 7 7 8 7 7 8 7 5 5 4 6 5 5 3 7	6 2

Problem L. Limited Swaps

Time limit: 2 seconds
Memory limit: 512 megabytes

Lina is playing with n cubes placed in a row. Each cube has an integer from 1 to n written on it. Every integer from 1 to n appears on exactly one cube.

Initially, the numbers on the cubes from left to right are $a_1, a_2, \ldots, a_n$. Lina wants the numbers on the cubes from left to right to be $b_1, b_2, \ldots, b_n$.

Lina can swap any two adjacent cubes, but only if the difference between the numbers on them is at least 2. This operation can be performed at most 20 000 times.

Find any sequence of swaps that transforms the initial configuration of numbers on the cubes into the desired one, or report that it is impossible.

Input

The first line contains a single integer n — the number of cubes ($1 \le n \le 100$).

The second line contains n distinct integers $a_1, a_2, \ldots, a_n$ — the initial numbers on the cubes from left to right ($1 \le a_i \le n$).

The third line contains n distinct integers $b_1, b_2, \ldots, b_n$ — the desired numbers on the cubes from left to right ($1 \le b_i \le n$).

Output

If it is impossible to obtain the desired configuration of numbers on the cubes from the initial one, print a single integer -1.

Otherwise, in the first line, print a single integer k — the number of swaps in your sequence ($0 \le k \le 20\,000$).

In the second line, print k integers $s_1, s_2, \ldots, s_k$ describing the operations in order ($1 \le s_i \le n - 1$). Integer s_i stands for "swap the s_i-th cube from the left with the $(s_i + 1)$-th cube from the left".

You do not have to find the shortest solution. Any solution satisfying the constraints will be accepted.

Examples

standard input	standard output
5 1 3 5 2 4 3 5 1 4 2	5 2 1 2 4 1
4 1 2 3 4 4 3 2 1	-1

Note

In the first example test, the configuration of numbers changes as follows:

$$1\ \underline{3\ 5}\ 2\ 4 \ \to\ \underline{1\ 5}\ 3\ 2\ 4 \ \to\ 5\ \underline{1\ 3}\ 2\ 4 \ \to\ 5\ 3\ 1\ \underline{2\ 4} \ \to\ \underline{5\ 3}\ 1\ 4\ 2 \ \to\ 3\ 5\ 1\ 4\ 2$$

In the second example test, making even a single swap in the initial configuration is impossible.

Problem M. Mex and Cards

Time limit: 3 seconds
Memory limit: 512 megabytes

Mike enjoys playing with cards. Each card in his deck has a single integer value from 0 to $n - 1$ written on it. Initially the deck contains a_i cards with value i.

Today Mike is learning the concept of *mex*. The mex of a collection of integers is the smallest non-negative integer that does not belong to the collection. For instance, $\text{mex}(\{4, 1, 4, 12, 0, 7, 0, 0, 5\}) = 2$.

Mike will distribute all cards in his deck into non-empty piles. Each card must belong to exactly one pile. He will then find the mex of the card values in each pile and add them all together. Mike wants to find a distribution that maximizes this sum.

Moreover, a sequence of q modifications happens to the deck: sometimes a new card is added to the deck, while other times a card is removed from the deck. Mike wants to find the distribution with the maximum sum of mexes at every point in the sequence: before the first modification, and after the first i modifications for every $i = 1, 2, \ldots, q$.

Input

The first line contains a single integer n — the range of card values ($1 \le n \le 2 \cdot 10^5$).

The second line contains n integers $a_0, a_1, \ldots, a_{n-1}$ — the number of cards with value $0, 1, \ldots, n - 1$ in the deck initially ($0 \le a_i \le 10^6$).

The third line contains a single integer q — the number of deck modifications ($0 \le q \le 2 \cdot 10^5$).

The i-th of the next q lines contains two integers p_i and v_i, describing the i-th modification ($1 \le p_i \le 2$; $0 \le v_i < n$). If $p_i = 1$, a new card with value v_i is added to the deck. If $p_i = 2$, a card with value p_i is removed from the deck.

It is guaranteed that if $p_i = 2$, then the deck contains at least one card with value v_i right before the i-th modification.

Output

Print $q + 1$ integers — the maximum possible sum of mexes for some valid distribution of all cards into piles after the first $0, 1, \ldots, q$ modifications to the deck.

Example

standard input	standard output
5 2 1 3 0 2 6 1 0 1 1 2 4 1 3 2 1 2 1	4 5 7 7 9 7 3

Note

For the initial deck of the example test, one of the best distributions is to assign the cards with values 0 and 2 into one pile, the cards with values $0, 1, 2, 2, 4$ into another pile, and the card with value 4 into the third pile. The sum of mexes in this distribution is $\text{mex}(\{0, 2\}) + \text{mex}(\{0, 1, 2, 2, 4\}) + \text{mex}(\{4\}) = 1 + 3 + 0 = 4$.

Problem N. New Time

Time limit: 2 seconds
Memory limit: 512 megabytes

Nikolay has a digital clock that displays time in 24-hour format, showing two integers: hours (from 00 to 23) and minutes (from 00 to 59). For example, the clock can show `00:00`, `18:42`, or `23:59`.

The clock has two buttons that can be used for manual adjustment:

- Button A sets the clock forward by 1 minute. For example, `05:33` becomes `05:34`, `16:59` becomes `17:00`, and `23:59` becomes `00:00`.
- Button B sets the clock forward by 1 hour. For example, `01:42` becomes `02:42`, and `23:14` becomes `00:14`.

Nikolay has noticed that the time on his clock does not look right. He wants to adjust the clock to the correct time by pressing the buttons as few times as possible.

Find the smallest number of button presses needed to adjust the clock.

Input

The first line contains the time shown on the clock in the `hh:mm` format ($00 \le$ `hh` ≤ 23; $00 \le$ `mm` ≤ 59).

The second line contains the correct time in the same format.

Output

Print a single integer — the smallest number of button presses Nikolay needs to adjust the time on his clock.

Examples

standard input	standard output
11:57 12:00	3
09:09 21:21	24
19:44 08:50	19

Note

In the first example test, Nikolay can adjust the time by pressing button A three times.

In the second example test, Nikolay should press button A and button B 12 times each.