Problem A. Another GCD Problem

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

Consider an infinite multiset s containing elements $a_1, a_2, \ldots, a_n$, each in infinite quantity. Based on this, we define the function $f(k)$ as follows. For each sub-multiset s' of s with exactly k elements, calculate the sum of the elements in s'. The value of $f(k)$ is then the greatest common divisor (GCD) of all these sums. Formally,

$$f(k) = \gcd_{\substack{s' \subseteq s \\ |s'|=k}} \left(\sum_{x \in s'} x \right).$$

For example, for $a = [3, 6]$, we have

$$f(2) = \gcd(3+3,\ 3+6,\ 6+6) = 3.$$

Find the maximum value of $f(k)$ and the smallest k that achieves this maximum value. In particular, if the maximum value does not exist, print "`infinite`".

Input

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

The first line contains an integer n ($1 \le n \le 10^5$), representing the number of distinct numbers in S.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^{18}$), representing the elements in S.

The sum of n over all test cases is at most $4 \cdot 10^5$.

Output

For each test case, if the maximum value of $f(k)$ exists, output a line with two integers, representing the maximum value of $f(k)$ and the smallest k that achieves this maximum value.

Otherwise, output "`infinite`" (without quotes).

Examples

standard input	*standard output*
2 2 3 6 2 2 2	3 1 infinite
2 3 1 4 7 4 4 16 28 34	3 3 6 3

Note

For the first test case in the first example, it can be observed that regardless of the value of k, $f(k) = 3$.

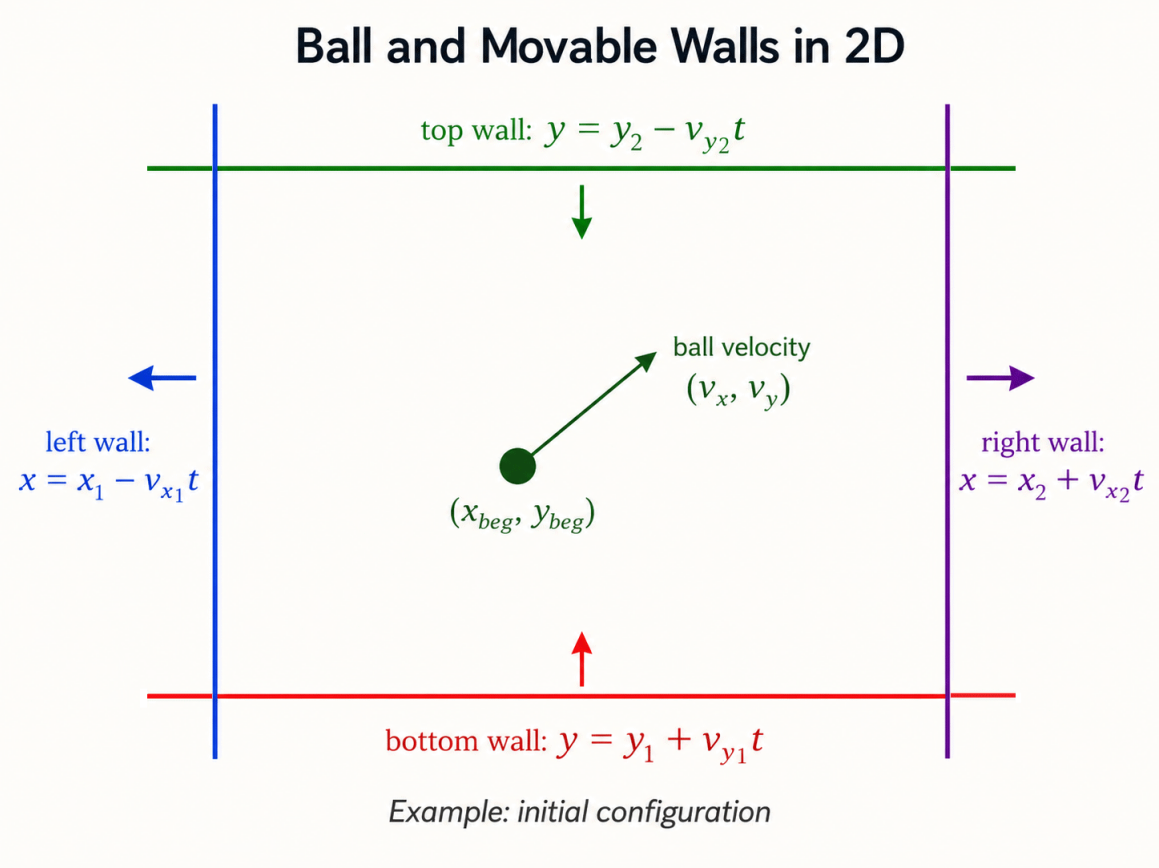
For the second test case in the first example, we have $f(k) = 2k$, thus f will grow infinitely and the maximum value does not exist.

Problem B. Ball And Movable Walls

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

In a two-dimensional experimental field, there are four infinitely long straight lines acting as movable "walls": the top and bottom are horizontal lines, while the left and right are vertical lines. We place a negligibly small ball between the four lines, and it starts moving in a uniform straight line. Whenever the ball touches one of the walls, it rebounds in the direction normal to that wall (that is, only the sign of the velocity in that direction changes), while the velocity in the other cardinal direction remains unchanged. All the while, the left and right vertical lines move away from each other, whereas the top and bottom horizontal lines move towards each other, eventually converging at some moment to trap the ball.

Given the initial position and velocity of the ball, as well as the initial positions and velocities of the four lines, you need to determine the coordinates of the ball at the moment the top and bottom lines meet.



Here is a formal description of the experiment.

The initial position of the ball is denoted as (x_{beg}, y_{beg}), and its initial velocity is denoted as (v_x, v_y).

The bottom horizontal line and the top horizontal line are initially located at $y = y_1$ and $y = y_2$, respectively. In time t, they move to $y = y_1 + v_{y_1}t$ and $y = y_2 - v_{y_2}t$.

The left vertical line and the right vertical line are initially located at $x = x_1$ and $x = x_2$, respectively. In time t, they move to $x = x_1 - v_{x_1}t$ and $x = x_2 + v_{x_2}t$.

It is guaranteed that the ball is initially strictly inside the rectangle formed by the four lines, that is, $x_1 < x_{beg} < x_2$ and $y_1 < y_{beg} < y_2$.

Each collision occurs when the ball is moving faster than the wall in the corresponding direction. We guarantee that by ensuring that the ball's vertical velocity is always greater than the speeds of the two

horizontal lines, that is, $v_{y_1} < |v_y|$ and $v_{y_2} < |v_y|$.

To ensure that the top and bottom lines will definitely meet, we guarantee that the speeds of the two horizontal lines are not both 0.

The specific rules for collisions are as follows:

1. The ball moves in a uniform straight line when not in contact with any line.
2. When in contact with a horizontal line, only the vertical velocity is inverted (changing v_y to $-v_y$), while the horizontal velocity remains unchanged.
3. When in contact with a vertical line, only the horizontal velocity is inverted (changing v_x to $-v_x$), while the vertical velocity remains unchanged.
4. The determination of rebounds is independent of the translation speeds of the lines and only involves "inverting" the corresponding velocity components.

Input

The first line of input contains an integer t ($1 \le t \le 100$), the number of test cases. For each test case:

The first line contains four integers: x_{beg}, y_{beg}, v_x, and v_y ($-20 \le v_x, v_y \le 20, v_x \neq 0, v_y \neq 0$), representing the initial coordinates and velocity of the ball.

The second line contains four integers: y_1, y_2, v_{y_1}, and v_{y_2} ($0 \le y_1 < y_{beg} < y_2 \le 10^6$, $0 \le v_{y_1}, v_{y_2} < |v_y|$, $v_{y_1} + v_{y_2} \neq 0$), representing the initial positions and velocities of the horizontal lines.

The third line contains four integers: x_1, x_2, v_{x_1}, and v_{x_2} ($0 \le x_1 < x_{beg} < x_2 \le 10^6$, $0 \le v_{x_1}, v_{x_2} \le 20$), representing the initial positions and velocities of the vertical lines.

Output

For each test case, output a line containing two space-separated real numbers x and y: the coordinates of the ball at the moment it stops.

Each of your answers will be considered correct if its absolute or relative error is at most 10^{-3}. In other words, if your answer is *out* and the true answer is *ans*, then your answer will be considered correct if

$$\frac{|out - ans|}{\max(1, |ans|)} \le 10^{-3}.$$

Examples

standard input	*standard output*
1 8 5 3 7 0 20 2 3 0 30 1 1	20 8
2 108469 765432 20 20 4 999999 1 0 108464 108470 1 0 108470 765432 20 20 4 999999 1 0 108465 108471 1 0	-0.0789012981284893 999999 0.921098701871511 999999

Problem C. Covering Problems

Input file:	*standard input*
Output file:	*standard output*
Time limit:	6 seconds
Memory limit:	1024 mebibytes

First, consider the *Sequence Covering Problem*. In this problem, three non-negative integer sequences of length n are given initially: a, b, and c. You can perform zero or more operations. In each operation, choose an interval $[\ell, r]$ at a cost of $b_\ell + c_r$, provided that the minimum value of $a_\ell, a_{\ell+1}, \ldots, a_r$ is not 0. Then decrease all values $a_\ell, a_{\ell+1}, \ldots, a_r$ by 1. The answer to the problem is the minimum possible cost to reduce all numbers in a to 0.

Then we consider the *Many Sequence Covering Problem*. In this problem, two non-negative integer sequences of length n are given initially: d and e. You can perform zero or more operations. In each operation, choose $i \in [1, n]$, and then either increase b_i by 1 at a cost of d_i, or increase c_i by 1 at a cost of e_i. After all operations are completed, solve the Sequence Covering Problem for the modified sequences b and c, denoting the answer as p and the total cost as q. The answer to the problem is the maximum possible value of $p - q$, or "`INF`" if this value can be made arbitrarily large.

But your goal is to solve the *Many Many Sequence Covering Problem*. In this problem, you are given five sequences of length n, which are a, b, c, d, and e. What's more, the sequences can be only partially determined! For each i, if a_i is non-negative, then it is the given value of a_i; otherwise, the true value of a_i could be any integer in the range $[0, -a_i]$. The same is true for the sequences b, c, d, and e. You need to calculate the sum of answers corresponding to the Many Sequence Covering Problem for all possible cases. Since the answer may be `INF`, you need to separately provide the sum of all answers when they are not `INF`, as well as the count of cases where the answer is `INF`. Since the answers may be large, output the results modulo 998 244 353.

Input

The first line of the input contains an integer n ($1 \le n \le 5000$), the length of the sequences.

Then follow five lines containing n integers each: $a_1, \ldots, a_n$ are given on the first of these lines, $b_1, \ldots, b_n$ on the second, $c_1, \ldots, c_n$ on the third, $d_1, \ldots, d_n$ on the fourth, and $e_1, \ldots, e_n$ on the fifth of them ($-5000 \le a_i, b_i, c_i, d_i, e_i \le 5000$).

Output

Output a single line containing two integers: the sum of all answers modulo 998 244 353 when the answer is not `INF`, and the count of cases where the answer is `INF`, also modulo 998 244 353.

Examples

standard input	*standard output*
2 1 1 1 2 2 1 -1 -1 -1 -1	8 12
3 -1 -2 2 1 3 0 -3 0 -2 1 -3 0 1 3 3	408 228

Problem D. Director and Excitement

Input file:	*standard input*
Output file:	*standard output*
Time limit:	5 seconds
Memory limit:	1024 mebibytes

The Cook Chicken Potato Contest is one of the most renowned events in the culinary world. The competition venue provides k cooking stations, and before the start, the participants will be divided into k equally sized teams by the Contest Director, known as Small Q.

To test the teamwork among participants and to add more excitement to the competition, Small Q wants to arrange the teams in such a way that the strength differences among participants in the same team are as large as possible. Let the strengths of the participants be denoted as $a_1, a_2, \ldots, a_n$, and their respective team assignments as $t_1, t_2, \ldots, t_n$. Small Q defines the excitement level of the competition as:

$$D = \min_{1 \le i < j \le n} \begin{cases} |a_i - a_j|, & t_i = t_j \\ +\infty, & t_i \neq t_j \end{cases}$$

Now, there are n potential participants. The possible strength of the i-th participant is described by an interval $[\ell_i, r_i]$, indicating that their actual strength during a certain competition could turn out to be any real number within that interval. The participants are sorted by strength in non-decreasing order, which means that, for all $1 \le i < j \le n$, we have $\ell_i \le \ell_j$ and $r_i \le r_j$.

Small Q has q competition plans, where the i-th plan is to invite participants with indices between L_i and R_i. You need to help Small Q determine whether he can arrange the teams in such a way that it would be possible for the excitement level of the competition to be at least D_i.

Input

The first line of input contains an integer t ($1 \le t \le 10^5$), the number of test cases. For each test case:

The first line contains two integers, n and k ($1 \le n \le 5 \cdot 10^5$, $1 \le k \le \min(5, n)$), representing the number of potential participants and the number of teams.

The next n lines contain two integers, ℓ_i and r_i ($0 \le \ell_i \le r_i \le 10^{12}$) for the i-th participant, indicating the possible strength they can exhibit. For all $1 \le i < n$, we have $\ell_i \le \ell_{i+1}$ and $r_i \le r_{i+1}$.

The next line contains an integer q ($1 \le q \le 10^5$), indicating the number of competition plans.

The next q lines contain three integers: L_i, R_i, and D_i ($1 \le L_i \le R_i \le n$, $(R_i - L_i + 1)$ is divisible by k, and $0 \le D_i \le 10^{12}$), indicating that the i-th competition plan is to invite participants with indices from L_i to R_i, and Small Q expects the excitement level to be at least D_i.

The sum of n over all test cases is at most 10^6. The sum of q over all test cases is at most 10^5.

Output

For each test case, output q lines. For the i-th output, print "`YES`" or "`NO`" to indicate whether Small Q's expectation for the i-th plan can possibly be met.

Example

standard input	*standard output*
2 4 2 1 1 3 3 4 4 6 6 3 1 2 3 3 4 2 1 4 2 5 1 1 3 2 3 4 6 7 10 8 12 6 1 3 2 1 3 3 2 4 4 2 4 5 3 5 4 3 5 5	YES YES YES YES NO YES NO YES NO

Problem E. Fun With Bitwise OR

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

You are given a sequence $a_1, a_2, \ldots, a_n$ of length n that contains only integers 1 and 2. You can perform the following operation zero or more times: choose a position i such that $1 \le i < n$, delete a_i and a_{i+1} from the sequence, and insert the value $a_i \mid a_{i+1}$ in their place, where $\mid$ denotes the bitwise OR. Note that, after each operation, the length of the sequence will decrease by 1.

For example, if the sequence is $a = [1, 2, 1]$ and you choose to perform the operation on $i = 2$, the sequence will change to $a = [1, 3]$.

Determine how many different sequences can be produced after performing zero or more operations, and output the result modulo $10^9 + 7$. Two sequences are considered different if and only if their lengths differ or if they have different elements at any particular position.

As n may be large, the sequence will be input in a compressed format where identical numbers are grouped together. It is guaranteed that the lengths of each segment of identical numbers are non-decreasing from front to back.

Input

The first line of input contains an integer t ($1 \le t \le 10^6$), the number of test cases. For each test case:

The first line contains two integers, m and a_1 ($1 \le m \le 10^6$, $1 \le a_1 \le 2$) representing the number of segments in the sequence and the value of a_1.

The second line contains m integers $\ell_1, \ell_2, \ldots, \ell_m$ ($1 \le \ell_1 \le \ell_2 \le \ldots \le \ell_m \le 10^9$), where ℓ_i represents the length of the i-th segment in the sequence.

Since the values in adjacent segments are different, the sequence of length $n = \sum_{i=1}^{m} \ell_i$ can be uniquely determined by a_1 and $\ell_1, \ell_2, \ldots, \ell_m$.

The sum of m over all test cases is at most 10^6.

Output

For each test case, output an integer representing the answer modulo $10^9 + 7$.

Example

standard input	*standard output*
2 3 1 1 1 2 8 2 1 2 3 4 5 6 7 8	7 2961300

Note

In the example, the first test case represents the sequence $a = [1, 2, 1, 1]$. The different sequences that can appear after performing zero or more operations are: $[1, 2, 1]$, $[1, 2, 1, 1]$, $[1, 3]$, $[1, 3, 1]$, $[3]$, $[3, 1]$, $[3, 1, 1]$.

Problem F. Great Snowfall

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	1024 mebibytes

After a heavy snowfall, Little W needs to clean up his yard to make the uneven snow piles look at least a bit tidier. Little W's yard can be viewed as an $n \times m$ grid, where the cell in the i-th row and j-th column is covered with snow of relative height $h_{i,j}$, which can be positive, negative, or zero. Little W can perform the following operations to clear the snow piles:

1. Choose $1 \le i < n$, $1 \le j \le m$, and push the snow pile from the i-th row and the j-th column to the next row. This operation will decrease $h_{i,j}$ by 1 and increase $h_{i+1,j}$ by 1. This operation has no cost.
2. Choose $1 \le i \le n$, $1 \le j < m$, and push the snow pile from the i-th row and the j-th column to the next column. This operation will decrease $h_{i,j}$ by 1 and increase $h_{i,j+1}$ by 1. This operation has no cost.
3. Choose $1 \le i \le n$, $1 \le j \le m$, and add snow to the snow pile in the i-th row and the j-th column. This operation will increase $h_{i,j}$ by 1. This operation costs 1.
4. Choose $1 \le i \le n$, $1 \le j \le m$, and remove snow from the snow pile in the i-th row and the j-th column. This operation will decrease $h_{i,j}$ by 1. This operation costs 1.

Little W wants to perform zero or more operations to make all $h_{i,j}$ equal to 0, while minimizing the total cost. Can you help him calculate the minimum cost?

Input

The first line of input contains an integer t ($1 \le t \le 10^6$), the number of test cases. For each test case:

The first line contains two integers, n and m ($1 \le n, m \le 1000$), the number of rows and columns in the yard.

The next n lines describe the yard. The i-th of these lines contains m integers $h_{i,1}, h_{i,2}, \ldots, h_{i,m}$ ($-10^9 \le h_{i,j} \le 10^9$), where the j-th integer represents the relative height of snow in the i-th row and the j-th column.

The sum of $n \cdot m$ over all test cases does not exceed 10^6.

Output

For each test case, output a single line with an integer: the minimum cost required for Little W to achieve his goal.

Example

standard input	*standard output*
3 1 1 5 1 2 1 -1 2 2 -1 0 1 1	5 0 3

Problem G. Efficient Summation

Input file:	*standard input*
Output file:	*standard output*
Time limit:	13 seconds
Memory limit:	1024 mebibytes

Let us denote the prime factorization of an integer y as:

$$y = \prod_{i=1}^{k} p_i^{\alpha_i}, \text{ where } p_1 < p_2 < \ldots < p_k.$$

Now, given a sequence r of length m that does not contain duplicate elements, define $f(y)$ as:

$$f(y) = \prod_{i=1}^{m} p_{r_i} \cdot \alpha_{r_i}.$$

If $r_i > k$, consider $p_{r_i} \cdot \alpha_{r_i} = 1$.

You are given q queries. Each query provides an integer x in the form of its prime factorization. Your task is to compute the value of

$$\sum_{i=1}^{\lfloor \frac{n}{x} \rfloor} f(i \cdot x) \bmod 2^{32}.$$

To reduce the output size, print the XOR sum of all answers.

Input

The first line of input contains three integers: n, m, and q ($1 \le n \le 7 \cdot 10^8$, $1 \le m \le 25$, $1 \le q \le 5 \cdot 10^5$), representing the upper limit for queries, the length of the sequence r, and the number of queries, respectively.

The second line contains m integers $r_1, r_2, \ldots, r_m$ ($1 \le r_i \le 25$, all r_i are distinct).

The next q lines describe queries. Each query is given on a single line which starts with an integer ℓ indicating the number of terms in the prime factorization of x. It is followed by 2ℓ integers $p_1, \alpha_1, p_2, \alpha_2, \ldots, p_\ell, \alpha_\ell$ (p_i are distinct primes in ascending order, $\alpha_i \ge 1$). This describes the value

$$x = \prod_{i=1}^{\ell} p_i^{\alpha_i}$$

such that $1 \le x \le n$. Specifically, $\ell = 0$ indicates that $x = 1$.

Output

Output a single line containing one integer: the XOR sum of all query answers.

Example

standard input	*standard output*
10 2 5 2 3 0 1 5 1 1 2 1 1 7 1 2 2 1 3 1	31

Problem H. How Many Rounds Are Expected?

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

There are n sets, and the i-th set contains a_i elements, with a total of $2m$ distinct elements ($\sum a_i = 2m$), where each element belongs to a unique set.

In each round of operations, we randomly match all elements into m pairs. For each matched pair, we randomly select one element, remove it from its original set, and place it into the set of the element it is matched with. The question is: what is the expected number of rounds until all elements belong to a single set?

Output the answer modulo 998 244 353.

Input

The first line of input contains an integer t ($1 \le t \le 100$), the number of test cases. For each test case:

The first line contains two integers n and m ($1 \le n \le 2m \le 400$), representing the number of initial sets and the number of matches per round.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 2 \cdot m$, $\sum a_i = 2 \cdot m$), where a_i indicates that the i-th set has a_i elements.

The sum of n over all test cases is at most 800. The sum of m over all test cases is at most 400.

Output

For each test case, output a single line containing an integer: the answer modulo 998 244 353. Formally, the answer can be represented as a rational number p/q, and you have to print the value r such that $p \equiv q \cdot r \pmod{998\,244\,353}$.

Examples

standard input	*standard output*
4 2 3 2 4 2 3 3 3 3 3 1 2 3 4 3 1 1 2 2	5 499122182 748683271 7
3 6 5 4 2 1 1 1 1 7 7 1 4 2 3 2 1 1 8 10 6 3 1 4 2 2 1 1	341357698 460332729 534908689

Note

Explanation for the first example:

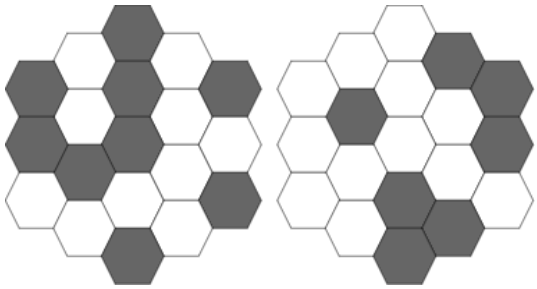
For each of the four test cases in this example, there are 6 elements. In one round, there are 15 possible matching ways, and for each matching there are $2^3 = 8$ ways to choose one element from every matched pair. Thus, there are 120 equally likely operations in one round. Let E_S denote the expected number of remaining rounds from state S.

- For the first test case: We represent the initial state as $[2, 4]$. Among the 120 operations, 24 operations result in the state $[6]$, while the remaining 96 operations keep the state $[2, 4]$. Therefore, $E_{[2,4]} = 1 + \frac{96}{120} E_{[2,4]} = 5$.
- For the second test case: We represent the initial state as $[3, 3]$. Among the 120 operations, 12 operations result in the state $[6]$, while the remaining 108 operations result in the state $[2, 4]$. Therefore, the expected value is $1 + \frac{108}{120} \cdot 5 = \frac{11}{2}$, which is 499 122 182 modulo 998 244 353.
- For the third test case: We represent the initial state as $[1, 2, 3]$. First, consider the auxiliary state $[2, 2, 2]$: among the 120 operations, 72 operations result in $[2, 4]$, while 48 operations keep the state $[2, 2, 2]$. Hence, $E_{[2,2,2]} = 1 + \frac{72}{120} \cdot 5 + \frac{48}{120} E_{[2,2,2]} = \frac{20}{3}$. For the state $[1, 2, 3]$, among the 120 operations, 6 operations result in $[6]$, 78 operations result in $[2, 4]$, and 36 operations result in $[2, 2, 2]$. Therefore, the expected value is $1 + \frac{78}{120} \cdot 5 + \frac{36}{120} \cdot \frac{20}{3} = \frac{25}{4}$, which is 748 683 271 modulo 998 244 353.
- For the fourth test case: We represent the initial state as $[1, 1, 2, 2]$. Among the 120 operations, 48 operations result in $[2, 4]$, while the remaining 72 operations result in $[2, 2, 2]$. Therefore, the expected value is $1 + \frac{48}{120} \cdot 5 + \frac{72}{120} \cdot \frac{20}{3} = 7$.

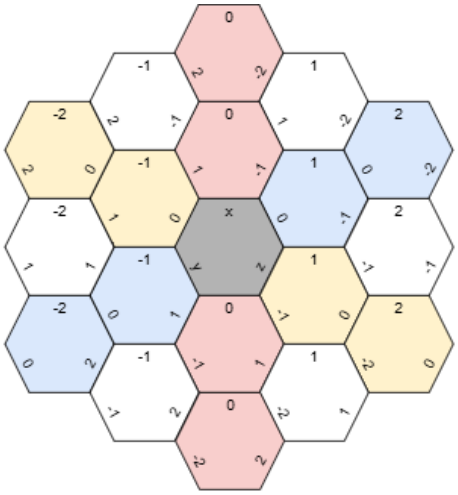
Problem I. Infinite Hexagonal Grids

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

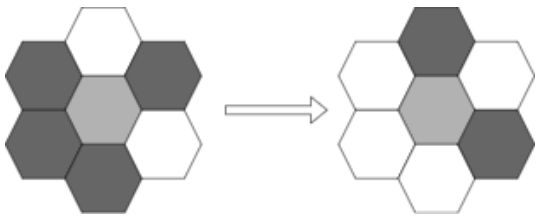
You are given two infinite hexagonal grids, where some grid cells are black and some grid cells are white. Two such grids are shown in the figure below (they correspond to the first example):



We describe each grid cell in the grid using a three-dimensional coordinate (x, y, z) such that $x, y, z \in \mathbb{Z}$ and $x + y + z = 0$, as illustrated in the figure below:



We can perform a flip operation: select a three-dimensional coordinate (x, y, z) such that $x, y, z \in \mathbb{Z}$ and $x + y + z = 0$, and then flip the colors of the grid cells surrounding that coordinate: black flips to white, and white flips to black. Specifically, we flip the colors of the grid cells $(x, y - 1, z + 1)$, $(x + 1, y - 1, z)$, $(x + 1, y, z - 1)$, $(x, y + 1, z - 1)$, $(x - 1, y + 1, z)$, and $(x - 1, y, z + 1)$. An example is shown in the figure below:



The question is whether it is possible to perform several flip operations on the first hexagonal grid so that the color of each grid cell matches that of the second hexagonal grid.

Input

The first line of input contains an integer t ($1 \le t \le 100$), the number of test cases. For each test case:

The first line contains two integers, n and m ($1 \le n, m \le 10^5$), representing the number of black grid cells on the two hexagonal grids.

The next n lines contain the coordinates of distinct black cells on the first hexagonal grid, where the i-th line contains three integers: x_i, y_i, and z_i ($-10^9 \le x_i, y_i, z_i \le 10^9$, $x_i + y_i + z_i = 0$).

The following m lines contain the coordinates of distinct black cells on the second hexagonal grid, where the i-th line contains three integers: u_i, v_i, and w_i ($-10^9 \le u_i, v_i, w_i \le 10^9$, $u_i + v_i + w_i = 0$).

The sum of n over all test cases is at most $2 \cdot 10^5$. The sum of m over all test cases is at most $2 \cdot 10^5$.

Output

For each test case, output "`YES`" if it is possible to perform several flip operations on the first hexagonal grid so that the color of each grid cell matches that of the second hexagonal grid; otherwise, output "`NO`".

Examples

standard input	*standard output*
1 9 7 0 2 -2 -2 2 0 0 1 -1 2 0 -2 -1 0 1 2 -2 0 0 -2 2 0 0 0 -2 1 1 -1 1 0 1 1 -2 2 0 -2 2 -1 -1 0 -1 1 0 -2 2 1 -2 1	YES
2 4 2 -1 1 0 -1 0 1 0 -1 1 1 0 -1 0 1 -1 1 -1 0 4 3 -1 1 0 -1 0 1 0 -1 1 1 0 -1 0 0 0 0 1 -1 1 -1 0	YES NO

Problem J. Judge of Gensokyo

Input file:	*standard input*
Output file:	*standard output*
Time limit:	4 seconds
Memory limit:	1024 mebibytes

Image source: Bad Apple!! PV [Shadow Picture]

Shiki is the judge of Gensokyo and often needs to take notes with pen and paper. Shiki has discovered that many English letters, such as `ovw`, form ligatures when written. For example, two consecutive `vs` written together look like a `w`; consecutive `v` and `w` also connect, for instance, `wvvwvwv` looks like a string of length 10 of `v`s.

Shiki considers a string to be good if and only if it is completely symmetrical when written on paper. For example, `wvowv` is symmetrical because it has three sharp angles, one round shape, and three sharp angles when written; however, `vowow` is not symmetrical because it has two sharp angles on the left and three on the right.

Now Shiki has given you a string s that she recorded, which consists only of the characters "`ovw`". You need to find the longest substring of s such that the substring is good.

Input

The first line of input contains an integer t ($1 \le t \le 2 \cdot 10^6$), the number of test cases. For each test case:

The first line contains an integer n ($1 \le n \le 10^7$), representing the length of the string s.

The second line contains a string s of length n, which consists only of the characters "`ovw`".

The sum of n over all test cases is at most 10^7.

Output

For each test case, output a line containing the longest good substring of s.

If there are multiple solutions, output any one of them.

Example

standard input	*standard output*
3 9 vwwwowvvw 12 voooooooowwv 12 wwvovoovvvvv	wwwowvvw oooooooo vvvvv

Problem K. Knapsack And Greedy Algorithm

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

The 01-knapsack problem is a classic combinatorial optimization problem in algorithmic competitions. Little W has learned a greedy algorithm to solve this problem. The 01-knapsack problem is defined as follows.

Consider n items: their weights are positive integers $w_1, w_2, \ldots, w_n$, and their values are positive integers $v_1, v_2, \ldots, v_n$. We have a knapsack with capacity c. We need to find $x_1, x_2, \ldots, x_n$, each of them either 0 or 1, such that the sum of weights $\sum_{i=1}^{n} w_i \cdot x_i \le c$ and the sum of values $v_t = \sum_{i=1}^{n} v_i \cdot x_i$ is the maximum possible.

Little W's greedy algorithm works as follows.

Sort the n items in descending order based on the value-to-weight ratio v_i/w_i; when ratios are the same, sort by decreasing w_i. Initialize a variable c_0 to 0, and iterate i from 1 to n. If $c_0 + w_i \le c$, set $x_i \leftarrow 1$ and $c_0 \leftarrow c_0 + w_i$; otherwise, set $x_i \leftarrow 0$. After iterating, you will obtain the desired $x_1, x_2, \ldots, x_n$ and v_t.

You certainly know that this algorithm is incorrect, but Little W does not believe it. Even when you provide Little W with some counterexamples, Little W still believes that this algorithm can yield the optimal v_t for many different values of c. Therefore, provided with a capacity limit c_{lim}, you now wish to construct a list of weights $w_1, w_2, \ldots, w_n$ and a list of values $v_1, v_2, \ldots, v_n$ such that:

1. For all c such that $2 \le c \le c_{lim}$, Little W's algorithm cannot achieve the optimal v_t.
2. Under condition 1, n is as small as possible.
3. Under conditions 1 and 2, $\max(w_1, w_2, \ldots, w_n)$ is as small as possible.
4. Under conditions 1, 2, and 3, $\max(v_1, v_2, \ldots, v_n)$ is as small as possible.

Now you need to construct a set that satisfies the above conditions for the 01-knapsack to convince Little W. Can you do it? If there are multiple construction methods, you can output any one of them.

Input

The input consists of a single line containing an integer c_{lim} ($2 \le c_{lim} \le 5000$), the upper bound for c.

Output

The output should contain the following:

The first line should contain an integer n ($1 \le n \le 10^4$), the number of items in the constructed 01-knapsack.

The second line should contain n integers $w_1, w_2, \ldots, w_n$ ($1 \le w_i \le c_{lim}$), the weights of the items.

The third line should contain n integers $v_1, v_2, \ldots, v_n$ ($1 \le v_i \le 10^9$), the values of the items.

It can be proven that, given the problem and input conditions, a solution satisfying the above data ranges can always be found.

Example

standard input	*standard output*
2	2 1 2 2 3

Problem L. Let's Avoid Obstacles

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

Consider an $n \times m$ grid with row numbers ranging from 1 to n and column numbers ranging from 1 to m. Each column, except for the leftmost column 1 and the rightmost column m, can have at most one obstacle; the leftmost and rightmost columns are guaranteed to have no obstacles.

You start from any cell in the leftmost column (you can choose the row), and your goal is to reach any cell in the rightmost column (you can choose the row). Suppose you are currently at row i and column j. You can choose one of the following three operations at each step:

- Move right: from (i, j) to $(i, j + 1)$;
- Move up: from (i, j) to $(i - 1, j)$;
- Move down: from (i, j) to $(i + 1, j)$.

You are not allowed to move left, and at any time, you cannot enter an obstacle cell or move outside the grid.

There are k obstacles (numbered from $i = 0$ to $k - 1$ after sorting by column number). For each obstacle, you must choose to "go around from above" or "go around from below".

If the i-th obstacle is at row r_i and column c_i:

- If you choose "above", your row number in column c_i must always be less than r_i;
- If you choose "below", your row number in column c_i must always be greater than r_i.

The choices for different columns are independent, resulting in a total of 2^k possible schemes.

Your task: for each scheme, calculate the minimum number of steps from any row in the leftmost column to any row in the rightmost column while satisfying all the constraints of that scheme; if there is no feasible path under that scheme, output -1.

Input

The first line of input contains an integer t ($1 \le t \le 50$), the number of test cases. For each test case:

The first line contains two integers, n and m ($1 \le n \le 100$, $2 \le m \le 100$), the number of rows and columns in the grid.

The second line contains an integer k ($0 \le k \le \min(m - 2, 10)$), the number of obstacles.

The next k lines describe obstacles. The i-th of them contains two integers, r_i and c_i ($1 \le r_i \le n$, $1 < c_1 < c_2 < \ldots < c_k < m$), indicating that there is an obstacle at row r_i and column c_i.

The sum of $n \cdot m$ over all test cases is at most 5000. There is no such constraint on the sum of k over all test cases.

Output

For each test case, output a line containing 2^k integers, representing the minimum number of steps for each scheme numbered from 0 to $2^k - 1$. Adjacent numbers should be separated by a space. If a certain scheme has no solution, output -1.

Scheme numbering explanation:

- Each scheme can be viewed as a binary number of length k.
- The binary digits, from the least significant bit to the most significant bit, correspond to the obstacles $0, 1, \ldots, k - 1$.

- A value 0 means going around the obstacle from above; a value 1 means going around from below.

For example, if $k = 3$, there are $2^3 = 8$ schemes corresponding to the following table:

Scheme #	Binary Representation (Least → Most Significant Bit)	Bypass Choice (for obstacles 0, 1, 2)
0	000	Above, Above, Above
1	100	Below, Above, Above
2	010	Above, Below, Above
3	110	Below, Below, Above
4	001	Above, Above, Below
5	101	Below, Above, Below
6	011	Above, Below, Below
7	111	Below, Below, Below

The program should output the minimum number of steps for each scheme in ascending order of scheme numbers.

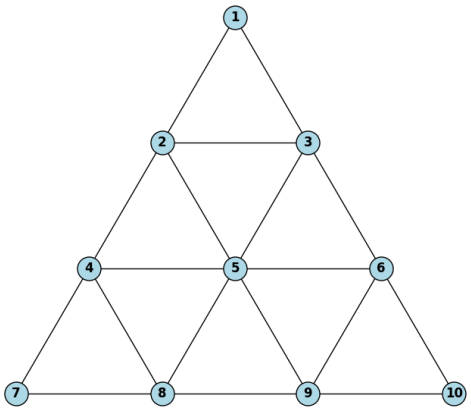
Example

standard input	*standard output*
3 3 6 2 3 4 2 5 3 4 1 1 2 3 6 2 3 2 1 5	5 -1 -1 -1 -1 3 -1 -1 5 -1

Problem M. Multiple Triangular Intersections

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

In Town A, there are $(n+1)(n+2)/2$ intersections, which are connected by $3n$ *paths*, forming an equilateral triangle with a side length of n *roads*. The case for $n = 3$ is shown in the figure below:



These $3n$ paths can be divided into three orientations: left diagonal, horizontal, and right diagonal, each containing n paths. The i-th path in each orientation consists of i roads.

For example, when $n = 3$:

- The left diagonal paths from the smallest to the largest number of roads are $(6 \leftrightarrow 9)$, $(3 \leftrightarrow 5 \leftrightarrow 8)$, $(1 \leftrightarrow 2 \leftrightarrow 4 \leftrightarrow 7)$.
- The horizontal paths from the smallest to the largest number of roads are $(2 \leftrightarrow 3)$, $(4 \leftrightarrow 5 \leftrightarrow 6)$, $(7 \leftrightarrow 8 \leftrightarrow 9 \leftrightarrow 10)$.
- The right diagonal paths from the smallest to the largest number of roads are $(4 \leftrightarrow 8)$, $(2 \leftrightarrow 5 \leftrightarrow 9)$, $(1 \leftrightarrow 3 \leftrightarrow 6 \leftrightarrow 10)$.

We call three distinct intersections (u, v, w) an *equilateral triangle intersection* of positive integer length ℓ if and only if we can satisfy the following after any permutation of the order of u, v, w:

- Starting from u, reach v by passing through ℓ left diagonal roads.
- Starting from v, reach w by passing through ℓ horizontal roads.
- Starting from w, reach u by passing through ℓ right diagonal roads.

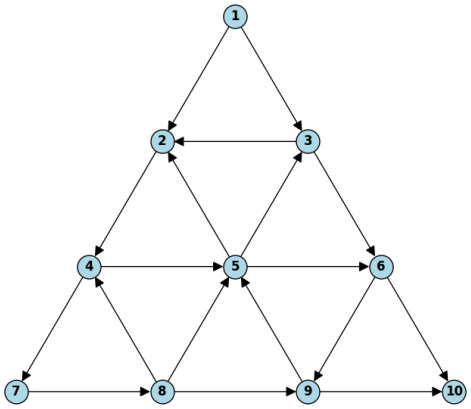
Or:

- Starting from u, reach v by passing through ℓ right diagonal roads.
- Starting from v, reach w by passing through ℓ horizontal roads.
- Starting from w, reach u by passing through ℓ left diagonal roads.

For example, in the figure above, $(2, 4, 5)$ and $(2, 3, 5)$ are equilateral triangle intersections, while $(2, 3, 4)$ is not.

To address traffic congestion, Town A has decided to make each road directed. After the direction is set, each road has a unique fixed direction, and the roads on each path have the same direction.

The following figure shows one possible direction (corresponding to the third test case of the first example):



We call three distinct intersections (u, v, w) a *connected equilateral triangle intersection* if and only if they form an equilateral triangle with sides of positive integer length ℓ in the graph, and they can reach each other through the 3ℓ roads that make up the equilateral triangle. For example, in the figure above:

- $(2, 4, 5)$ is a connected equilateral triangle intersection because they are not only an equilateral triangle intersection but can also reach each other only through the roads $(2 \to 4,\ 4 \to 5,\ 5 \to 2)$ that make up that equilateral triangle intersection.
- $(2, 3, 4)$ is not a connected equilateral triangle intersection because they are not an equilateral triangle intersection.
- $(2, 3, 5)$ is not a connected equilateral triangle intersection because they are an equilateral triangle intersection but cannot reach each other through the **roads** $(5 \to 3,\ 3 \to 2,\ 2 \leftarrow 5)$ that make up that equilateral triangle intersection.

Now, given the directions of all directed paths, you are asked how many connected equilateral triangle intersections exist in this graph.

Input

The first line of input contains an integer t $(1 \le t \le 10^5)$, the number of test cases. For each test case:

The first line contains an integer n $(1 \le n \le 10^5)$, the length of the equilateral triangle.

The second line contains a binary string s_1 of length n, indicating that if $s_{1,i} = \texttt{0}$, the direction of the left diagonal path consisting of i roads is from the upper right to the lower left, and the reverse if $s_{1,i} = \texttt{1}$.

The third line contains a binary string s_2 of length n, indicating that if $s_{2,i} = \texttt{0}$, the direction of the horizontal path consisting of i roads is from left to right, and the reverse if $s_{2,i} = \texttt{1}$.

The fourth line contains a string s_3 of length n, indicating that if $s_{3,i} = \texttt{0}$, the direction of the right diagonal path consisting of i roads is from the lower right to the upper left, and the reverse if $s_{3,i} = \texttt{1}$.

The sum of n over all test cases is at most 10^5.

Output

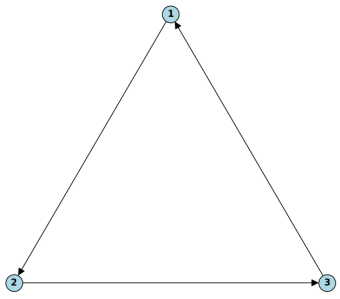
For each test case, output an integer indicating the number of connected equilateral triangle intersections.

Examples

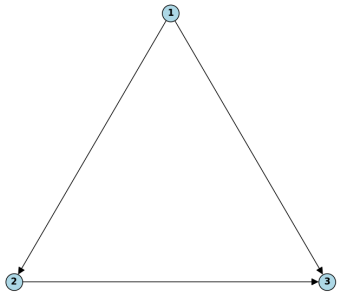
standard input	*standard output*
3 1 0 0 0 1 0 0 1 3 010 100 001	1 0 4
1 4 0011 1100 0011	6

Note

In the first example, the figure for the first test case is:



The figure for the second test case is:



The connected equilateral triangle intersections are:

- $(2, 4, 5)$;
- $(4, 7, 8)$;
- $(5, 6, 9)$;
- $(2, 7, 9)$.