Problem A. Square Kingdom

In the Square Kingdom, n residents numbered from 1 to n live alone on top of a tall stone pillar. The height of the pillar for the i-th resident is $\left(i + \frac{b}{a}\right)^2$ units above the ground.

Since everyone lives so high up, the only way to visit a neighbor is by using a ladder. The kingdom builds one ladder for every pair of residents. The length of each ladder is exactly the absolute difference in height between the two pillars it connects.

With $\frac{n(n-1)}{2}$ ladders in total, it becomes hard to manage them all. You are asked for the length of the k-th ladder in ascending order of length. This information will help the kingdom plan repairs, deliveries, and community events.

Input

The only line contains four integers n ($2 \le n \le 10^{12}$), k ($1 \le k \le \min\left\{\frac{n(n-1)}{2}, 10^{12}\right\}$), a ($1 \le a \le 10^6$), and b ($0 \le b \le 10^{12}$).

Output

Output a line containing two integers p and q, denoting that the length of the k-th ladder in ascending order of length can be represented as $\frac{p}{q}$ with two integers p and q ($p \ge 0$, $q \ge 1$, $\gcd(p, q) = 1$), where $\gcd(p, q)$ is the greatest common divisor of p and q.

Examples

standard input
3 1 3 1
standard output
11 3

standard input
3 2 3 1
standard output
17 3

standard input
3 3 3 1
standard output
28 3

standard input
1414215 1000000000000 1000000 1000000000000
standard output
4823373069559 1

This page is intentionally left blank

Problem B. Buggy Painting Software I

Sulfox the fennec fox is a coder who loves creating digital art in his spare time. Worn out from a day of debugging, he opens his painting software as usual, only to find the software itself is also riddled with bugs after the latest update, one major issue being that it stutters noticeably whenever he edits a pixel.

The painting software treats the canvas as an ordered stack of layers. Each layer is a grid of pixels with n rows and m columns, each pixel holding a *visible value* in $\{0, 1, \ldots, nm\}$, where 0 denotes transparent, and positive values are identifiers for nm distinct colors. Since layers may occlude one another, at any position, the visible value of the composite image equals the visible value of the topmost non-transparent pixel among all layers; if all layers are transparent at that position, the visible value is 0.

Sulfox already knows exactly what he wants to draw: a target image denoted by a matrix $(p_{i,j})_{n\times m}$ of visible values. To finish as quickly as possible, he wants to minimize the total "lag cost" caused by the software bug. Starting with no layers, Sulfox may perform the following operations any number of times:

- **Create Layer**: insert a new layer on top for free and initialize it either as a constant color layer with every pixel set to the same chosen color c ($1 \le c \le nm$), or as an empty layer with every pixel transparent;
- **Brush Tool**: spend a to set any single pixel of any layer to any chosen color c ($1 \le c \le nm$);
- **Eraser Tool**: spend b to make any single pixel of any layer transparent.

Help Sulfox determine the minimum total cost needed to perform operations so that the composite image matches the target image, i.e., the visible values at corresponding positions of both images are equal.

Input

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

The first line contains four integers n, m ($1 \le n, m \le 500$), a, and b ($1 \le a, b \le 10^6$), denoting the image dimensions, the cost of using the Brush Tool, and the cost of using the Eraser Tool, respectively.

The i-th line of the next n lines contains m integers $p_{i,1}, p_{i,2}, \ldots, p_{i,m}$ ($0 \le p_{i,j} \le nm$), where $p_{i,j}$ denotes the visible value of the target image at the i-th row and the j-th column.

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

Output

For each test case, output one line containing an integer, representing the minimum total cost.

Example

standard input	standard output
3 1 2 3 2 0 1 2 2 1 1 1 0 2 3 3 3 5 3 2 4 4 4 1 4 4 4 2	2 3 11

Note

For the first test case of the sample case, Sulfox can first create a layer with color 1, and then spend $b = 2$ to make the pixel at the first row and the first column transparent.

Problem C. Buggy Painting Software II

This is a communication problem.

Well... after surviving the nightmarish stutter in **Problem B. Buggy Painting Software I**, Sulfox soon faces a far more serious bug — his painting software saves any color image in black-and-white format.

Here, a color image supports m distinct colors labeled 1 through m, and can be viewed as a sequence of length $3m$ composed of color labels, where each color appears **exactly three times**. A black-and-white image supports only black (labeled 0) and white (labeled 1), and can be viewed as a binary sequence. When saving a color image, the software chooses a non-empty bipartition (S_0, S_1) of $S = \{1, 2, \ldots, m\}$ (so $S_0 \cup S_1 = S$, $S_0 \cap S_1 = \varnothing$, and both S_0 and S_1 are non-empty) and converts the original sequence into a binary sequence of the same length by mapping every color in S_0 to 0 and every color in S_1 to 1. The resulting binary sequence will be the saved black-and-white image.

To outsmart this bug, Sulfox wants to create n color images with respective integer feature values $x_1, x_2, \ldots, x_n$ (values may repeat) ranging from 1 to m. Your task is to help Sulfox design these color images so that each image's feature remains identifiable even after the color loss. More specifically, the i-th color image ($1 \le i \le n$) must satisfy that, no matter how the non-empty bipartition (S_0, S_1) is chosen, its feature value x_i can always be recovered from the saved black-and-white image.

To verify the correctness of your design, your program will be **run twice** on each test: first for constructing n color images, and then for recovering the original feature values from n black-and-white images. The second run will receive exactly the n binary sequences mapped from each of the n sequences you output in the first run by independently chosen non-empty bipartitions. Apart from that, the second run will receive no additional information from the first run. Your solution passes a test if you correctly recover the corresponding feature value from each black-and-white image.

Note that the applied bipartitions are unpredictable and may be **adaptive** to your design, and different bipartitions may apply to the color images with the same feature value.

Input

The first line contains three integers op ($op \in \{1, 2\}$), n, and m ($2 \le n, m \le 5 \times 10^5$, $nm \le 10^6$), denoting which run of the program it is, the number of color images, and the number of distinct colors, respectively. It is guaranteed that n and m remain consistent across both runs.

If $op = 1$, your program must construct the color images. The next n lines follow, where the i-th line contains a single integer x_i ($1 \le x_i \le m$), representing the feature value for the i-th image to be designed.

If $op = 2$, your program must recover the feature values. The next n lines follow, each describing a black-and-white image. Each line contains $3m$ space-separated integers of 0 or 1, forming a binary sequence that results from one of your designed color images. These n sequences are given in an **arbitrary order** and do not necessarily correspond to the order in which the feature values were given in the first run.

Output

If $op = 1$, output n lines, where the i-th line describes the designed color image for the i-th feature value x_i from the input. Each line contains $3m$ space-separated integers. The sequence on each line must be a valid color image, i.e., a sequence where each integer from 1 to m appears exactly three times.

If $op = 2$, output n lines. For each of the n binary sequences given in the input, output a single line containing one integer, representing the original feature value recovered from that image. The order of your output must correspond to the order of the binary sequences in the input.

Examples

standard input	standard output
1 2 3 1 2	1 1 1 2 2 2 3 3 3 1 2 3 1 2 3 1 2 3
2 2 3 1 1 0 1 1 0 1 1 0 0 0 0 1 1 1 0 0 0	2 1

Note

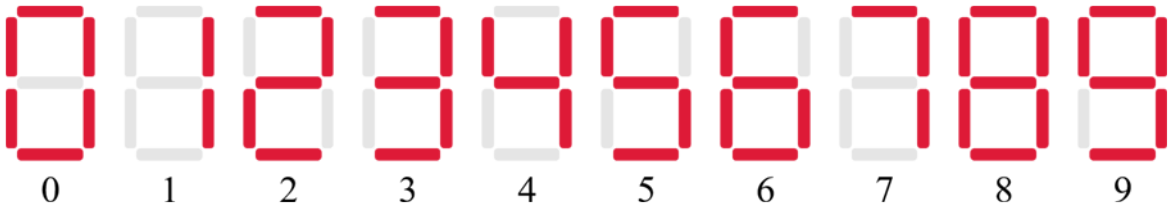
The example shows two runs of a certain solution on the sample case.

In the first run, a simple (and incorrect for general cases) strategy is used: feature value 1 is designed as consecutive blocks of three identical colors, and feature value 2 is designed as a repeating pattern of three distinct colors.

In the second run, the first binary sequence (from bipartition $(S_0 = \{3\}, S_1 = \{1, 2\})$) is identified by its repeating pattern of three to recover feature value 2, while the second binary sequence (from bipartition $(S_0 = \{1, 3\}, S_1 = \{2\})$) is identified by its consecutive blocks of three to recover feature value 1. Since the judge provides these sequences in a swapped order, the output is 2 and then 1.

Problem D. LED Display Renovation

You have an LED display capable of showing an n-digit integer, where each digit is represented using the standard 7-segment layout.



Initially, each of the 7 digit segments in every digit block may be in one of three states:

- Functional (denoted by '`W`'): behaves correctly, i.e., it lights up or turns off as required by the digit being displayed;
- Stuck ON (denoted by '`1`'): always lit regardless of the digit being displayed;
- Stuck OFF (denoted by '`0`'): always unlit regardless of the digit being displayed.

You can renovate at most k digit segments, i.e., convert at most k digit segments into functional. You need to maximize the number of integers that can be correctly displayed and find the number of ways to achieve the maximum. Note that you can renovate digit segments that are initially functional, and each digit segment can be renovated at most once. Two ways of renovation are considered different if there is at least one digit segment that is renovated in one way but not in the other.

An integer is correctly displayed if and only if:

- It has no leading zeros, unless the integer is exactly 0;
- It is right-aligned on the display: if the integer has d ($1 \le d \le n$) digits, then only the rightmost d digit blocks are used, and the leftmost $(n - d)$ blocks should be left blank and show nothing;
- For each of the d used digit blocks, all 7 digit segments match the target digit's display pattern when functional segments show the correct state for that digit and faulty segments remain stuck in their state.

Input

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

The first line contains two integers n ($1 \le n \le 9$) and k ($0 < k < 7n$), denoting the number of digits the LED display can show and the number of digit segments you can repair.

The next 7 lines each contain a string of exactly $(5n - 1)$ characters, describing the initial state of the LED display in the form of an ASCII image.

The display consists of n digit blocks, each occupying 4 columns, separated by single-column gaps. All digit segments are represented by two identical characters of '`W`', '`0`', or '`1`', and all other positions are filled with the character '`.`' that are purely for formatting. See the sample input and the notes for details.

Output

For each test case, output a line containing two integers, indicating the maximum number of integers that can be correctly displayed by renovating at most k digit segments, as well as the number of ways to achieve the maximum.

Example

standard input
2 3 2 .00...00...00. 0..0.0..0.0..1 0..0.0..0.0..1 .00...00...00. 0..W.0..0.0..1 0..W.0..0.0..1 .00...00...00. 9 62 .WW...WW...WW...WW...WW...WW...WW...WW...WW. W..W.W..W.W..W.W..W.W..W.W..W.W..W.W..W.W..W W..W.W..W.W..W.W..W.W..W.W..W.W..W.W..W.W..W .WW...WW...WW...WW...WW...WW...WW...WW...WW. W..W.W..W.W..W.W..W.W..W.W..W.W..W.W..W.W..W W..W.W..W.W..W.W..W.W..W.W..W.W..W.W..W.W..W .WW...WW...WW...WW...WW...WW...WW...WW...WW.
standard output
2 23 1000000000 9223372036854775807

Note

In each digit block of the LED display in the form of an ASCII image, the digit segments are as follows:

Segment	Part	Positions (row, column)
0	Top horizontal	$(1,2)$, $(1,3)$
1	Upper-left vertical	$(2,1)$, $(3,1)$
2	Upper-right vertical	$(2,4)$, $(3,4)$
3	Middle horizontal	$(4,2)$, $(4,3)$
4	Lower-left vertical	$(5,1)$, $(6,1)$
5	Lower-right vertical	$(5,4)$, $(6,4)$
6	Bottom horizontal	$(7,2)$, $(7,3)$

The display patterns for digits $0, 1, 2, \ldots, 9$ (`1` = lit, `0` = unlit) are as follows:

Digit	Pattern (segments $0, 1, 2, \ldots, 6$)	Explanation
0	1110111	All except middle
1	0010010	Upper-right and lower-right
2	1011101	Top, upper-right, middle, lower-left, and bottom
3	1011011	Top, upper-right, middle, lower-right, and bottom
4	0111010	Upper-left, upper-right, middle, and lower-right
5	1101011	Top, upper-left, middle, lower-right, and bottom
6	1101111	All except upper-right
7	1010010	Top, upper-right, and lower-right
8	1111111	All segments
9	1111011	All except lower-left

Problem E. Play It by Ear

You are playing a popular card game. There are $2n$ distinct cards labeled 1 through $2n$, shuffled uniformly at random and piled up into a deck. You draw the top n cards as your starting hand, and the remaining deck order is **hidden** from you. Each turn, you choose any one card from your hand, play it by placing it at the bottom of the deck, and then draw the current top card, so your hand size remains n.

You have m quests to complete in order: the i-th quest requires you to play the card labeled a_i, and it becomes active only after the $(i-1)$-th quest is completed (unless it is the first quest, which is initially active). Playing the card labeled a_i before the i-th quest is active **does not count** toward the i-th quest. You know any of $a_1, a_2, \ldots, a_m$ (values may repeat) in advance.

Assuming you play cards under the optimal pure strategy, compute the minimum expected number of turns needed to complete all m quests over the random initial shuffle, modulo 998 244 353.

Formally, a *pure strategy* means that at each turn, the card chosen depends only on the observable history, and identical known information (initial hand, respective played and drawn cards in past turns, and quest completion status) deterministically yields the same chosen card. For a pure strategy σ and an initial shuffle π, let $f_\sigma(\pi)$ be the number of turns needed to complete all m quests when following σ; the value to be computed equals $(\min_\sigma E[f_\sigma(\pi)]) \bmod 998\,244\,353$, where the expectation $E[\cdot]$ is taken over the uniform distribution on all $(2n)!$ possible shuffles of the initial deck.

Input

The first line of the input contains an integer T ($1 \le T \le 1\,000$), denoting the number of test cases. For each test case:

The first line contains two integers n and m ($2 \le n \le 5\,000$, $1 \le m \le 2 \times 10^5$).

The second line contains m integers $a_1, a_2, \ldots, a_m$ ($1 \le a_i \le 2n$).

It is guaranteed that the sum of n over all test cases does not exceed 5 000, and the sum of m over all test cases does not exceed 2×10^5.

Output

For each test case, since the minimum expected number of turns can be denoted by an irreducible fraction $\frac{p}{q}$, you only need to print an integer r in one line satisfying $0 \le r < 998\,244\,353$ and $r \cdot q \equiv p \pmod{998\,244\,353}$. It can be proved that such r exists and is unique.

Example

standard input	standard output
3 2 1 1 3 7 2 5 1 2 6 2 1 1000 2 1116 1116	249561090 299473317 748684517

Note

For the first test case of the sample case, an optimal pure strategy is: if the card labeled 1 is in your initial hand, play it immediately; otherwise repeatedly play any card until you draw it and play it in your next turn. Hence the minimum expected number of turns is $\frac{1}{2} \times 1 + \frac{1}{4} \times 2 + \frac{1}{4} \times 3 = \frac{7}{4}$.

This page is intentionally left blank

Problem F. The Bond Beyond Time

Sulfox the fennec fox has built a maze and invited his two friends, Alice and Bob, to try it. The maze can be abstracted as a simple, connected undirected graph with n vertices and m edges, where Alice and Bob start at two distinct specified vertices.

Before any player moves, Sulfox must choose a direction for every undirected edge, turning the maze into a directed graph. Then, in an attempt to find each other, both players move in rounds within the oriented maze. In each round, they act simultaneously and independently, neither leaving marks nor communicating; rather, they adopt the "most brainless" strategy:

- If a player is currently at a vertex that has one or more outgoing edges, they arbitrarily traverse an outgoing edge to the adjacent vertex;
- otherwise, if their current vertex has no outgoing edges, they remain at that vertex.

Sulfox, being proud of his maze-design skills, wants to orient the passages so that Alice and Bob can never meet. Given the undirected graph and the starting vertices of Alice and Bob, please help Sulfox orient every edge so that no matter how Alice and Bob make their choices in every round, they will never occupy the same vertex at the end of any round. If such orientation does not exist, report it.

Input

The first line of the input contains an integer T ($1 \le T \le 1\,000$), denoting the number of test cases. For each test case:

The first line contains four integers n ($2 \le n \le 300$), m ($n - 1 \le m \le \frac{n(n-1)}{2}$), x, and y ($1 \le x, y \le n$, $x \ne y$), denoting the number of vertices and edges in the maze, and the starting vertices of Alice and Bob, respectively.

Each of the next m lines contains two integers u and v ($1 \le u, v \le n$), describing an undirected edge connecting vertices u and v. It is guaranteed that the given graph is connected and contains no self-loops or multiple edges.

It is guaranteed that there is at most 1 test case with n greater than 30.

Output

For each test case:

If it is possible to orient all edges so that Alice and Bob can never occupy the same vertex at the end of any round, print "`Yes`" (without quotes) in the first line.

Then output m lines, each containing two integers u' and v' ($1 \le u', v' \le n$), describing a directed edge from u' to v'. The unordered pair (u', v') must correspond to an existing undirected edge in the input. The edges may be printed in any order.

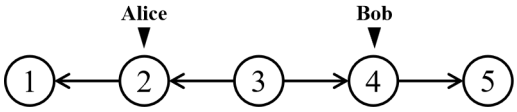
If no valid orientation exists, output a single line containing "`No`" (without quotes).

Example

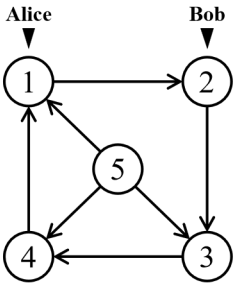
standard input	standard output
3 5 4 2 4 1 2 2 3 3 4 4 5 5 7 1 2 1 2 2 3 3 4 4 5 1 5 3 5 1 4 2 1 1 2 1 2	Yes 2 1 3 2 3 4 4 5 Yes 1 2 2 3 3 4 4 1 5 1 5 3 5 4 No

Note

For the first test case of the sample case, a valid orientation is illustrated as follows. In the first round, Alice must move from vertex 2 to vertex 1, while Bob must move from vertex 4 to vertex 5. After this round, both players are at vertices with no outgoing edges and will remain there forever, thus never meeting.



For the second test case of the sample case, a valid orientation is illustrated as follows. Both players are confined to the cycle $1 \to 2 \to 3 \to 4 \to 1$. Since they both advance one step along the cycle in each round, Alice will always remain one step behind Bob, thus never meeting.



For the third test case of the sample case, with only one edge connecting the two vertices, directing it $1 \to 2$ forces Alice to move to Bob's vertex, whereas directing it $2 \to 1$ forces Bob to move to Alice's vertex. Since one player always joins the stationary player, a meeting is unavoidable regardless of the orientation.

Problem G. Collision Damage

In a 2D physics-based video game, two characters P and Q move across the battlefield. Each character is represented by a convex polygon that serves as its collision box. When the two collision boxes overlap, the game calculates the area of their intersection as the collision damage.

However, due to unpredictable wind currents in the arena, character Q can be displaced by an arbitrary translation vector $\mathbf{t} = (t_x, t_y)$, but it cannot rotate or flip. You need to calculate the expected collision damage when Q is uniformly and randomly placed such that it actually collides with P, i.e., their intersection has a positive area.

More formally, define $f(\mathbf{t})$ as the area of the intersection between the polygon representing character P and the polygon representing character Q after being translated along $\mathbf{t}$. Let $D \subseteq \mathbb{R}^2$ be the set of all translation vectors $\mathbf{t}$ for which $f(\mathbf{t}) > 0$. It can be shown that D has a positive area, denoted by $|D|$. You need to calculate $\frac{1}{|D|} \iint_D f(\mathbf{t})\, d\mathbf{t}$.

Input

The first line of the input contains an integer T ($1 \le T \le 300$), denoting the number of test cases. For each test case:

- The first line contains two integers n and m ($3 \le n, m \le 1\,000$), denoting the numbers of vertices of the two convex polygons, respectively.
- Then n lines follow, each containing two integers x and y ($-10^4 \le x, y \le 10^4$), giving the vertex coordinates (x, y) of the polygon representing character P.
- Then m lines follow, each containing two integers x and y ($-10^4 \le x, y \le 10^4$), giving the vertex coordinates (x, y) of the polygon representing character Q.
- It is guaranteed that the vertices of a convex polygon are given in counter-clockwise order, and no three of them are collinear.

It is guaranteed that both the sum of n and the sum of m over all test cases do not exceed $1\,000$.

Output

For each test case, output a line containing a real number, representing the expected collision damage.

Your answer is acceptable if its absolute or relative error does not exceed 10^{-6}. Formally speaking, suppose that your output is a and the jury's answer is b, and your output is accepted if and only if $\frac{|a-b|}{\max(1,|b|)} \le 10^{-6}$.

Example

standard input	standard output
2 3 3 0 0 1 0 0 1 0 0 1 0 0 1 3 3 0 0 1 0 0 1 0 1 1 0 1 1	0.083333333333 0.125000000000

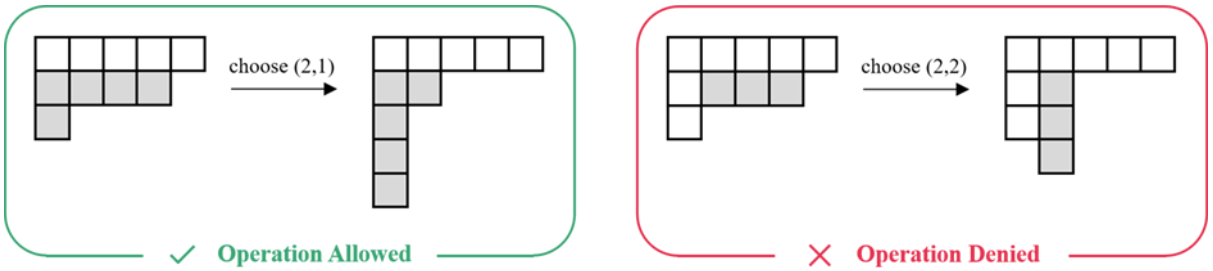
Problem H. Cute Young Diagram Counting

A *Young diagram* is a finite set of cells, arranged in left-aligned rows, with the row lengths in non-increasing order. It can be uniquely represented by an unordered integer partition $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_r)$ satisfying $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r \geq 1$, where r corresponds to the number of rows of the Young diagram, and λ_i corresponds to the number of cells in the i-th row for $i = 1, 2, \ldots, r$.

The *conjugate* of a Young diagram λ is another Young diagram obtained by transposing rows and columns, represented by the conjugate partition λ^T. More specifically, if λ has $c = \lambda_1$ columns, then $\lambda^T = (\mu_1, \mu_2, \ldots, \mu_c)$ where $\mu_j = |\{\, i : \lambda_i \geq j \,\}|$ for $j = 1, 2, \ldots, c$.

For a cell at the i-th row and the j-th column that belongs to λ, let the *subdiagram* anchored at (i, j) be the set of all cells (x, y) of λ with $x \geq i$ and $y \geq j$. It is easy to see that, this set is itself a Young diagram when viewed with (i, j) as its top-left corner.

Define the *cuteness* of a Young diagram λ as the number of distinct Young diagrams that can be obtained from λ by performing the following operation any number (possibly zero) of times: choose any cell (i, j) of the current Young diagram, take the subdiagram anchored at (i, j), and replace it with its conjugate anchored at the same position. The operation is allowed if and only if the overall set of cells still forms a valid Young diagram; otherwise the operation is denied and undone, as illustrated below.



You are given a non-increasing sequence of positive integers $a_1, a_2, \ldots, a_n$. For each $i = 1, 2, \ldots, n$, compute the cuteness of the Young diagram $\lambda^{(i)}$, modulo 998 244 353, where $\lambda^{(i)} = (a_1, a_2, \ldots, a_i)$.

Input

The first line contains a single integer n ($1 \leq n \leq 10^6$), denoting the length of the given sequence.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq n$). It is guaranteed that $a_1 \geq a_2 \geq \cdots \geq a_n$.

Output

Output n integers separated by spaces, where the i-th integer represents the cuteness of the Young diagram $\lambda^{(i)}$, modulo 998 244 353.

Example

standard input	standard output
3 3 2 1	2 3 1

Note

For the sample case:

- The distinct Young diagrams that can be obtained from $\lambda^{(1)}$ are (3) and $(1, 1, 1)$;
- The distinct Young diagrams that can be obtained from $\lambda^{(2)}$ are $(3, 2)$, $(3, 1, 1)$, and $(2, 2, 1)$;
- The only Young diagram that can be obtained from $\lambda^{(3)}$ is $(3, 2, 1)$ itself.

This page is intentionally left blank

Problem I. Volunteer Simulator

In an ICPC contest, there are multiple problems, and the contest lasts for a long duration. There are n accepted submissions in total, each representing a successful attempt by a team on a problem. When a team makes an accepted submission for a problem, we say that the team has solved that problem. Note that if a team submits multiple accepted solutions for the same problem, only the first such submission is considered as solving the problem for the first time.

You need to determine for each submission whether a balloon should be awarded according to the following rules:

- Before the scoreboard freeze (submission time < 240 minutes): when a team solves a problem for the first time, award a balloon corresponding to that problem.
- After the scoreboard freeze (submission time ≥ 240 minutes): when a team solves a problem for the first time, award a balloon corresponding to that problem if and only if the team has received **strictly fewer** than 3 balloons in total so far.

Input

The first line of input contains n ($1 \leq n \leq 5\,000$), denoting the number of accepted submissions.

The next n lines describe the submissions in chronological order. Each submission contains three integers a ($1 \leq a \leq 410$), b ($1 \leq b \leq 13$), and c ($0 \leq c < 300$), representing the team ID, the problem ID, and the submission time in minutes, respectively. It is guaranteed that the submission times are non-decreasing.

Output

For each submission, output the problem ID if a balloon corresponding to that problem should be awarded, or otherwise output 0.

Example

standard input	standard output
8 1 1 10 1 2 20 1 3 30 1 4 250 1 5 260 1 6 270 1 7 280 2 1 290	1 2 3 0 0 0 0 1

This page is intentionally left blank

Problem J. The Echoes of Chronos

In the year 3142, deep within the rings of Titan, the last archive of pre-Collapse human memory is sealed inside the Chronos Vault. Its entrance is guarded by the Echo Dial, a quantum-entangled lock composed of n chronal rings arranged in a single filament. Each ring displays a phase value in the range 0 to $m-1$, indicating its alignment with the cosmic rhythm. Initially, the i-th ring shows the value a_i.

The Echo Dial can only be adjusted through resonant tuning, governed by the following rules:

- In a single operation, you may choose any phase value $x \in [0, m)$ and simultaneously tune all rings currently displaying x either:
  - forward tuning: $x \to (x+1) \bmod m$, or
  - backward tuning: $x \to (x-1) \bmod m$.
- This operation affects every ring in the entire dial that currently resonates at phase x, regardless of position.

The Oracle of Echoes, the vault's sentient guardian, issues q diagnostic trials. Each trial is given as a query (l, r, v) and asks for the minimum number of resonant tuning operations required to set all rings in positions l through r (inclusive) to phase v.

All trials are hypothetical. The Echo Dial always starts from its original initial state for every query, and no trial alters the actual configuration of the dial. As the last Quantum Archivist, your duty is to answer all q queries efficiently.

Input

The first line contains three integers n, q ($1 \le n, q \le 2 \times 10^5$), and m ($1 \le m \le 10^9$).

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i < m$), denoting the initial phase of each ring.

Each of the next q lines contains three integers l, r ($1 \le l \le r \le n$) and v ($0 \le v < m$), describing a diagnostic trial.

Output

For each query, output a line containing an integer, denoting the minimum number of operations needed to set all rings in positions $[l, r]$ to phase v, assuming the dial starts from its original state and remains unchanged between queries.

Example

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

This page is intentionally left blank

Problem K. Relay Jump

There are n frogs numbered 1 through n, each initially located at an integer lattice point in the Cartesian plane. An external disturbance suddenly stimulates one of the frogs, triggering a chain reaction in which the frogs act according to the rules below.

When frog i ($1 \le i \le n$) is *stimulated*, it can choose to either:

- jump toward some other frog j ($1 \le j \le n$, $j \ne i$) and instantly land at the reflection of its current position across the point currently occupied by frog j (if frog i coincides with frog j before the jump, then it lands at the coinciding point); immediately after this jump, frog j becomes the next and only frog to be stimulated;
- or stay where it is, in which case all jumps end and no frog remains stimulated.

A frog may be stimulated multiple times at different moments, while some frogs may never be stimulated.

You observe that the first stimulated frog is numbered s, but there are too many jumps for you to capture the entire process. Fortunately, you recognize every frog, so you record both their initial positions $P_1, P_2, \ldots, P_n$ and their final positions $Q_1, Q_2, \ldots, Q_n$ after all jumps have ended. You also notice that all initial positions are **pairwise distinct**, and the final positions are **pairwise distinct** as well.

Let t be the index of the last stimulated frog, i.e., the frog that chooses to stay still when stimulated, after which no more jumps occur. Please find t.

Input

The first line contains two integers n ($2 \le n \le 10^5$) and s ($1 \le s \le n$), denoting the number of frogs and the index of the first stimulated frog, respectively.

The i-th of the next n lines contains four integers px_i, py_i, qx_i, and qy_i ($-10^9 \le px_i, py_i, qx_i, qy_i \le 10^9$), where $P_i = (px_i, py_i)$ and $Q_i = (qx_i, qy_i)$ represent the initial and final positions of frog i in the plane.

It is guaranteed that all initial positions $P_1, P_2, \ldots, P_n$ are pairwise distinct. All final positions $Q_1, Q_2, \ldots, Q_n$ are pairwise distinct as well, and correspond to some sequence of at most 2×10^5 stimulations following the described rules.

Output

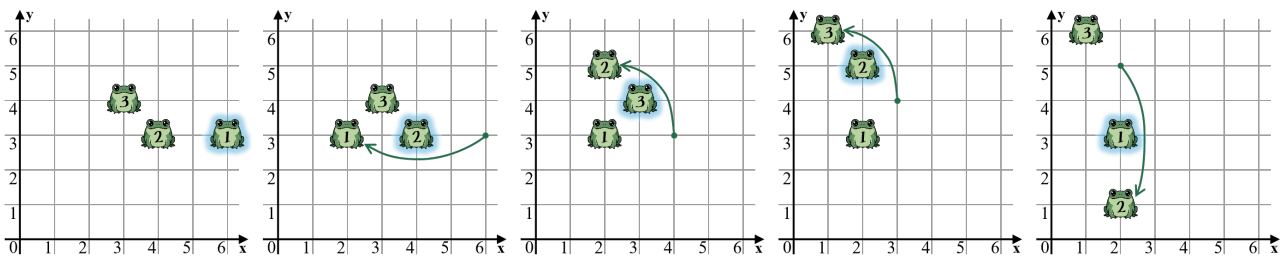
Output a single integer t, denoting the index of the frog that was last stimulated.

Example

standard input	standard output
3 1 6 3 2 3 4 3 2 1 3 4 1 6	1

Note

In the sample case, the corresponding sequence of stimulations is $[1, 2, 3, 2, 1]$.



This page is intentionally left blank

Problem L. Leo

In a future where digital logic has evolved beyond binary, a new system of signal processing, known as tri-color logic, has become the standard. This system defines four fundamental states in two categories:

- **Colored**: There are three colored states, denoted by R (Red), G (Green), and B (Blue), respectively. Each serves as a distinguishable "on" state, analogous to the traditional logic 1 signal.
- **Colorless**: The only colorless state is denoted by $*$ (Transparent). It serves as an "off" state, analogous to the traditional logic 0 signal.

Sulfox the fennec fox is designing a computing device based on tri-color logic technology, named Leo. It employs a simple combinational logic architecture, so we can regard it as a directed acyclic graph in which each node holds a state from the set $\{\mathrm{R}, \mathrm{G}, \mathrm{B}, *\}$. After fixing the machine's input scale to n, the graph is constructed as follows:

- Nodes $1, 2, \ldots, n$ are *input nodes*, whose states should be given as the input to the machine. Input nodes have no predecessors.
- Nodes $n+1, n+2, \ldots, n+m$ are *internal nodes*, whose states are computed based on the states of two predecessors, where the two predecessors of each internal node may be the same and are chosen from nodes with indices strictly less than its own. To ensure the machine's operating speed, m (the number of internal nodes) **must not exceed** $6n$.
- The internal node $n+m$ is also the only *output node*, whose state is the output of the machine.

Every internal node is of exactly one of the following two types, depending on how it combines the signals from its two predecessors:

- **Tri-Color AND**: If the states of its predecessors are identical, its own state will be that same state. If they differ, its own state will be $*$.
- **Tri-Color OR**: If the states of its predecessors are identical, its own state will be that same state. If one predecessor's state is $*$ while the other's is colored, its own state will be the colored one. If they differ and both are colored, its own state will be the colored state that is different from both.

With everything prepared, we now specify Leo's intended functionality. The states given to the n input nodes will be guaranteed to contain at least one occurrence of each of R, G, and B, where $*$ may appear any number (possibly zero) of times. For every such input, the state of the output node $n+m$ must be identical to **the third distinct colored state encountered** when examining the input nodes from node 1 to node n (ignoring all $*$). In other words, among R, G, and B it must output the colored state whose first occurrence has the largest index.

Given the machine's fixed input scale n, please complete the design of the internal nodes, i.e., specify the number of internal nodes and the type and predecessors of each internal node.

To verify the correctness of your design, for each test, the judge will generate $\left\lfloor \frac{10^7}{n} \right\rfloor$ valid input cases, each of which will be provided as your machine's input in turn. Your solution passes a test if your machine's output always meets Leo's intended functionality across all input cases.

Input

The only line contains an integer n ($3 \le n \le 10^5$), denoting the number of input nodes.

Output

In the first line, output an integer m ($1 \le m \le 6n$), denoting the number of internal nodes in your design.

Then output m lines, the i-th line containing a character t_i ($t_i \in \{$ '&' , '|' $\}$) and two integers u_i and v_i ($1 \le u_i, v_i < n + i$), denoting the type of the internal node $n + i$ and the indices of its two predecessors, where '&' represents the tri-color AND type and '|' represents the tri-color OR type.

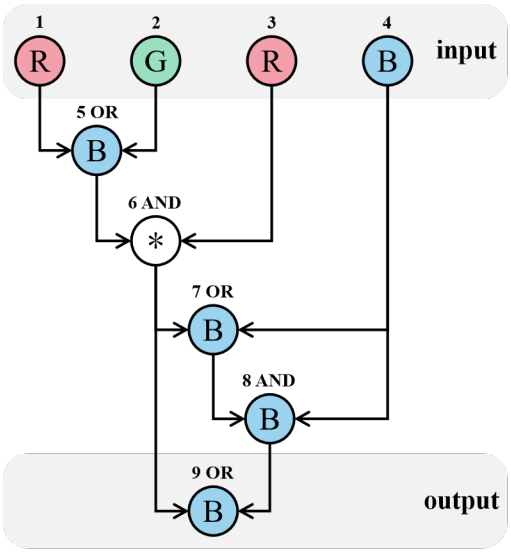
If there are multiple designs, you may output any of them. Note that you do not have to minimize m.

Example

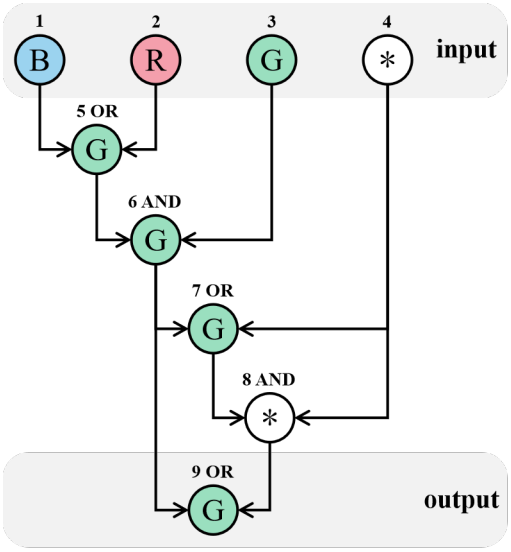
standard input	standard output
4	5 | 1 2 & 3 5 | 4 6 & 4 7 | 6 8

Note

It can be proven that the design in the sample case can always find the third encountered distinct colored state for every valid input of $n = 4$. For example, the figure below illustrates the machine's computation for the input "RGRB":



The figure below illustrates the machine's computation for the input "BRG∗":



Problem M. The End?

At the 2025 League of Legends Season 15 World Championship, T1 battled KT Rolster in a full five-game series, ultimately defeating KT with a 3-2 score to claim the championship. This match was exceptionally significant, giving birth to multiple new historical records.

As T1's star mid-laner, Faker has now achieved six World Championship titles in his personal career with this victory, further cementing his status as the "Greatest Player of All Time" in League of Legends. Besides Faker, his teammates Oner, Gumayusi, and Keria also accomplished the remarkable feat of securing three consecutive World Championship titles. Meanwhile, the newly joined top-laner Doran filled the gap in his personal accolades by winning his first World Championship.

The journey for the LPL region, however, was full of drama. The highly-touted first seed Bilibili Gaming (BLG) suffered a surprising upset loss to the North American team 100 Thieves in the very first round of the Swiss Stage. They were then eliminated in the crucial qualification match by Top Esports (TES), stopping at the Top 16 as a first seed and becoming one of the major upsets of the tournament. TES itself was swept 3-0 by T1 in the semi-finals, leaving the LPL region absent from the finals. Has the LPL reached its conclusion?

When we look back at the journey of the two finalist teams, everything began with the life-or-death knockout stage. The eight-team single-elimination bracket stipulated that teams were placed into the bracket based on their performance in the Swiss Stage, entering a series of head-to-head battles where the loser was immediately eliminated. The entire stage consisted of three rounds: the quarter-finals, where eight teams were narrowed down to four; the semi-finals, where these four teams produced two finalists; and the finals, where one ultimate champion was crowned from these two teams.

For Season 16, can our Team 1 win the championship? Please find the maximum probability of Team 1 winning the entire tournament.

More specifically, there are 8 teams participating in a single-elimination tournament. Each team i has two strength values a_i and b_i.

Before the tournament starts, you need to assign each team to a distinct seed from 1 to 8. The tournament follows the standard single-elimination bracket format:

- Round 1: seed 1 vs seed 2, seed 3 vs seed 4, seed 5 vs seed 6, seed 7 vs seed 8
- Round 2: winner of $(1, 2)$ vs winner of $(3, 4)$, winner of $(5, 6)$ vs winner of $(7, 8)$
- Round 3 (Final): winner of upper bracket $(1, 2, 3, 4)$ vs winner of lower bracket $(5, 6, 7, 8)$

When two teams compete against each other, the team with the smaller seed number uses its a-value as its strength, and the team with the larger seed number uses its b-value as its strength. If a team with strength x competes against a team with strength y, the probability that the first team wins is $\frac{x}{x+y}$.

You need to find the maximum probability of team 1 winning the entire tournament, considering all possible assignments of teams to seeds.

Input

The input consists of 8 lines. The i-th line contains two integers a_i and b_i ($1 \le a_i, b_i \le 100$), denoting the two strength values of team i.

Output

Output a line containing a real number, denoting the maximum probability that team 1 wins the tournament.

Your answer is acceptable if its absolute or relative error does not exceed 10^{-6}. Formally speaking, suppose that your output is a and the jury's answer is b, and your output is accepted if and only if $\frac{|a-b|}{\max(1,|b|)} \le 10^{-6}$.

Examples

standard input	standard output
10 80 20 70 30 60 40 50 50 40 60 30 70 20 80 10	0.329505822460368
100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100	0.125000000000000