A. Subsequence Permutation

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

A string s of length n, consisting of lowercase letters of the English alphabet, is given.

You must choose some number k between 0 and n. Then, you select k characters of s and permute them however you want. In this process, the positions of the other $n-k$ characters remain unchanged. You have to perform this operation exactly once.

For example, if $s =$ `"andrea"`, you can choose the $k = 4$ characters `"a_d_ea"` and permute them into `"d_e_aa"` so that after the operation the string becomes `"dneraa"`.

Determine the minimum k so that it is possible to sort s alphabetically (that is, after the operation its characters appear in alphabetical order).

Input

The first line contains a single integer t ($1 \le t \le 1000$) — the number of test cases. Then t test cases follow.

The first line of each test case contains one integer n ($1 \le n \le 40$) — the length of the string.

The second line of each test case contains the string s. It is guaranteed that s contains only lowercase letters of the English alphabet.

Output

For each test case, output the minimum k that allows you to obtain a string sorted alphabetically, through the operation described above.

Example

input
4 3 lol 10 codeforces 5 aaaaa 4 dcba

output
2 6 0 4

Note

In the **first test case**, we can choose the $k = 2$ characters `"_ol"` and rearrange them as `"_lo"` (so the resulting string is `"llo"`). It is not possible to sort the string choosing strictly less than 2 characters.

In the **second test case**, one possible way to sort s is to consider the $k = 6$ characters `"_o__force_"` and rearrange them as `"_c__efoor_"` (so the resulting string is `"ccdeefoors"`). One can show that it is not possible to sort the string choosing strictly less than 6 characters.

In the **third test case**, string s is already sorted (so we can choose $k = 0$ characters).

In the **fourth test case**, we can choose all $k = 4$ characters `"dcba"` and reverse the whole string (so the resulting string is `"abcd"`).

B. Running for Gold

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

The Olympic Games have just started and Federico is eager to watch the marathon race.

There will be n athletes, numbered from 1 to n, competing in the marathon, and all of them have taken part in 5 important

marathons, numbered from 1 to 5, in the past. For each $1 \le i \le n$ and $1 \le j \le 5$, Federico remembers that athlete i ranked $r_{i,j}$-th in marathon j (e.g., $r_{2,4} = 3$ means that athlete 2 was third in marathon 4).

Federico considers athlete x superior to athlete y if athlete x ranked better than athlete y in at least 3 past marathons, i.e., $r_{x,j} < r_{y,j}$ for at least 3 distinct values of j.

Federico believes that an athlete is likely to get the gold medal at the Olympics if he is superior to all other athletes.

Find any athlete who is likely to get the gold medal (that is, an athlete who is superior to all other athletes), or determine that there is no such athlete.

Input

The first line contains a single integer t ($1 \le t \le 1000$) — the number of test cases. Then t test cases follow.

The first line of each test case contains a single integer n ($1 \le n \le 50\,000$) — the number of athletes.

Then n lines follow, each describing the ranking positions of one athlete.

The i-th of these lines contains the 5 integers $r_{i,1}$, $r_{i,2}$, $r_{i,3}$, $r_{i,4}$, $r_{i,5}$ ($1 \le r_{i,j} \le 50\,000$) — the ranking positions of athlete i in the past 5 marathons. It is guaranteed that, in each of the 5 past marathons, the n athletes have distinct ranking positions, i.e., for each $1 \le j \le 5$, the n values $r_{1,j}$, $r_{2,j}$, $\ldots$, $r_{n,j}$ are distinct.

It is guaranteed that the sum of n over all test cases does not exceed $50\,000$.

Output

For each test case, print a single integer — the number of an athlete who is likely to get the gold medal (that is, an athlete who is superior to all other athletes). If there are no such athletes, print -1. If there is more than such one athlete, print any of them.

Example

input
4 1 50000 1 50000 50000 50000 3 10 10 20 30 30 20 20 30 10 10 30 30 10 20 20 3 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 6 9 5 3 7 1 7 4 1 6 8 5 6 7 3 2 6 7 8 8 6 4 2 2 4 5 8 3 6 9 4
output
1 -1 1 5

Note

Explanation of the first test case: There is only one athlete, therefore he is superior to everyone else (since there is no one else), and thus he is likely to get the gold medal.

Explanation of the second test case: There are $n = 3$ athletes.

- Athlete 1 is superior to athlete 2. Indeed athlete 1 ranks better than athlete 2 in the marathons 1, 2 and 3.
- Athlete 2 is superior to athlete 3. Indeed athlete 2 ranks better than athlete 3 in the marathons 1, 2, 4 and 5.
- Athlete 3 is superior to athlete 1. Indeed athlete 3 ranks better than athlete 1 in the marathons 3, 4 and 5.

Explanation of the third test case: There are $n = 3$ athletes.

- Athlete 1 is superior to athletes 2 and 3. Since he is superior to all other athletes, he is likely to get the gold medal.
- Athlete 2 is superior to athlete 3.
- Athlete 3 is not superior to any other athlete.

Explanation of the fourth test case: There are $n = 6$ athletes.

- Athlete 1 is superior to athletes 3, 4, 6.
- Athlete 2 is superior to athletes 1, 4, 6.
- Athlete 3 is superior to athletes 2, 4, 6.
- Athlete 4 is not superior to any other athlete.
- Athlete 5 is superior to athletes 1, 2, 3, 4, 6. Since he is superior to all other athletes, he is likely to get the gold medal.
- Athlete 6 is only superior to athlete 4.

C. Maximize the Intersections

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

On a circle lie $2n$ distinct points, with the following property: however you choose 3 chords that connect 3 disjoint pairs of points, no point strictly inside the circle belongs to all 3 chords. The points are numbered $1, 2, \ldots, 2n$ in clockwise order.

Initially, k chords connect k pairs of points, in such a way that all the $2k$ endpoints of these chords are distinct.

You want to draw $n - k$ additional chords that connect the remaining $2(n - k)$ points (each point must be an endpoint of exactly one chord).

In the end, let x be the total number of intersections among all n chords. Compute the maximum value that x can attain if you choose the $n - k$ chords optimally.

Note that the exact position of the $2n$ points is not relevant, as long as the property stated in the first paragraph holds.

Input

The first line contains a single integer t ($1 \le t \le 100$) — the number of test cases. Then t test cases follow.

The first line of each test case contains two integers n and k ($1 \le n \le 100$, $0 \le k \le n$) — half the number of points and the number of chords initially drawn.

Then k lines follow. The i-th of them contains two integers x_i and y_i ($1 \le x_i, y_i \le 2n$, $x_i \ne y_i$) — the endpoints of the i-th chord. It is guaranteed that the $2k$ numbers $x_1, y_1, x_2, y_2, \ldots, x_k, y_k$ are all distinct.

Output

For each test case, output the maximum number of intersections that can be obtained by drawing $n - k$ additional chords.

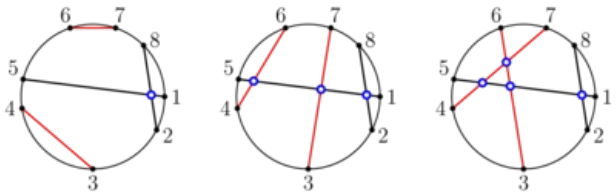
Example

input
4 4 2 8 2 1 5 1 1 2 1 2 0 10 6 14 6 2 20 9 10 13 18 15 12 11 7

output
4 0 1 14

Note

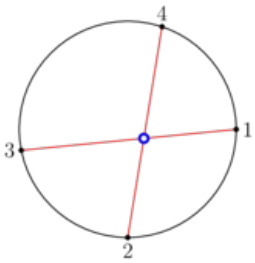
In the **first test case**, there are three ways to draw the 2 additional chords, shown below (black chords are the ones initially drawn, while red chords are the new ones):



We see that the third way gives the maximum number of intersections, namely 4.

In the **second test case**, there are no more chords to draw. Of course, with only one chord present there are no intersections.

In the **third test case**, we can make at most one intersection by drawing chords $1 - 3$ and $2 - 4$, as shown below:



D. Array Differentiation

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a sequence of n integers $a_1, a_2, \ldots, a_n$.

Does there exist a sequence of n integers $b_1, b_2, \ldots, b_n$ such that the following property holds?

- For each $1 \le i \le n$, there exist two (not necessarily distinct) indices j and k ($1 \le j, k \le n$) such that $a_i = b_j - b_k$.

Input

The first line contains a single integer t ($1 \le t \le 20$) — the number of test cases. Then t test cases follow.

The first line of each test case contains one integer n ($1 \le n \le 10$).

The second line of each test case contains the n integers $a_1, \ldots, a_n$ ($-10^5 \le a_i \le 10^5$).

Output

For each test case, output a line containing YES if a sequence $b_1, \ldots, b_n$ satisfying the required property exists, and NO otherwise.

Example

input
5 5 4 -7 -1 5 10 1 0 3 1 10 100 4 -3 2 10 2 9 25 -171 250 174 152 242 100 -205 -258
output
YES YES NO YES YES

Note

In the **first test case**, the sequence $b = [-9, 2, 1, 3, -2]$ satisfies the property. Indeed, the following holds:

- $a_1 = 4 = 2 - (-2) = b_2 - b_5$;
- $a_2 = -7 = -9 - (-2) = b_1 - b_5$;
- $a_3 = -1 = 1 - 2 = b_3 - b_2$;
- $a_4 = 5 = 3 - (-2) = b_4 - b_5$;
- $a_5 = 10 = 1 - (-9) = b_3 - b_1$.

In the **second test case**, it is sufficient to choose $b = [0]$, since $a_1 = 0 = 0 - 0 = b_1 - b_1$.

In the **third test case**, it is possible to show that no sequence b of length 3 satisfies the property.

E. Colors and Intervals

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

The numbers $1, 2, \ldots, n \cdot k$ are colored with n colors. These colors are indexed by $1, 2, \ldots, n$. For each $1 \le i \le n$, there are exactly k numbers colored with color i.

Let $[a, b]$ denote the interval of integers between a and b inclusive, that is, the set $\{a, a+1, \ldots, b\}$. You must choose n intervals $[a_1, b_1], [a_2, b_2], \ldots, [a_n, b_n]$ such that:

- for each $1 \le i \le n$, it holds $1 \le a_i < b_i \le n \cdot k$;
- for each $1 \le i \le n$, the numbers a_i and b_i are colored with color i;
- each number $1 \le x \le n \cdot k$ belongs to at most $\left\lceil \frac{n}{k-1} \right\rceil$ intervals.

One can show that such a family of intervals always exists under the given constraints.

Input

The first line contains two integers n and k ($1 \le n \le 100$, $2 \le k \le 100$) — the number of colors and the number of occurrences of

each color.

The second line contains $n \cdot k$ integers $c_1, c_2, \ldots, c_{nk}$ ($1 \le c_j \le n$), where c_j is the color of number j. It is guaranteed that, for each $1 \le i \le n$, it holds $c_j = i$ for exactly k distinct indices j.

Output

Output n lines. The i-th line should contain the two integers a_i and b_i.

If there are multiple valid choices of the intervals, output any.

Examples

input
4 3 2 4 3 1 1 4 2 3 2 1 3 4
output
4 5 1 7 8 11 6 12

input
1 2 1 1
output
1 2

input
3 3 3 1 2 3 2 1 2 1 3
output
6 8 3 7 1 4

input
2 3 2 1 1 1 2 2
output
2 3 5 6

Note

In the **first sample**, each number can be contained in at most $\left\lceil \frac{4}{3-1} \right\rceil = 2$ intervals. The output is described by the following picture:

In the **second sample**, the only interval to be chosen is forced to be $[1, 2]$, and each number is indeed contained in at most $\left\lceil \frac{1}{2-1} \right\rceil = 1$ interval.

In the **third sample**, each number can be contained in at most $\left\lceil \frac{3}{3-1} \right\rceil = 2$ intervals. The output is described by the following picture:

F. Telepanting

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

An ant moves on the real line with constant speed of 1 unit per second. It starts at 0 and always moves to the right (so its position increases by 1 each second).

There are n portals, the i-th of which is located at position x_i and teleports to position $y_i < x_i$. Each portal can be either *active* or *inactive*. The initial state of the i-th portal is determined by s_i: if $s_i = 0$ then the i-th portal is initially inactive, if $s_i = 1$ then the i-

th portal is initially active. When the ant travels through a portal (i.e., when its position coincides with the position of a portal):

- if the portal is inactive, it becomes active (in this case the path of the ant is not affected);
- if the portal is active, it becomes inactive and the ant is instantly teleported to the position y_i, where it keeps on moving as normal.

How long (from the instant it starts moving) does it take for the ant to reach the position $x_n + 1$? It can be shown that this happens in a finite amount of time. Since the answer may be very large, compute it modulo $998\,244\,353$.

Input

The first line contains the integer n ($1 \le n \le 2 \cdot 10^5$) — the number of portals.

The i-th of the next n lines contains three integers x_i, y_i and s_i ($1 \le y_i < x_i \le 10^9$, $s_i \in \{0, 1\}$) — the position of the i-th portal, the position where the ant is teleported when it travels through the i-th portal (if it is active), and the initial state of the i-th portal.

The positions of the portals are strictly increasing, that is $x_1 < x_2 < \cdots < x_n$. It is guaranteed that the $2n$ integers $x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n$ are all distinct.

Output

Output the amount of time elapsed, in seconds, from the instant the ant starts moving to the instant it reaches the position $x_n + 1$. Since the answer may be very large, output it modulo $998\,244\,353$.

Examples

input
4 3 2 0 6 5 1 7 4 0 8 1 1
output
23

input
1 454971987 406874902 1
output
503069073

input
5 243385510 42245605 0 644426565 574769163 0 708622105 208990040 0 786625660 616437691 0 899754846 382774619 0
output
899754847

input
5 200000000 100000000 1 600000000 400000000 0 800000000 300000000 0 900000000 700000000 1 1000000000 500000000 0
output
3511295

Note

Explanation of the first sample: The ant moves as follows (a curvy arrow denotes a teleporting, a straight arrow denotes normal movement with speed 1 and the time spent during the movement is written above the arrow).

$$0 \xrightarrow{6} 6 \rightsquigarrow 5 \xrightarrow{3} 8 \rightsquigarrow 1 \xrightarrow{2} 3 \rightsquigarrow 2 \xrightarrow{4} 6 \rightsquigarrow 5 \xrightarrow{2} 7 \rightsquigarrow 4 \xrightarrow{2} 6 \rightsquigarrow 5 \xrightarrow{4} 9$$

Notice that the total time is $6 + 3 + 2 + 4 + 2 + 2 + 4 = 23$.

Explanation of the second sample: The ant moves as follows (a curvy arrow denotes a teleporting, a straight arrow denotes normal movement with speed 1 and the time spent during the movement is written above the arrow).

$$0 \xrightarrow{454971987} 454971987 \rightsquigarrow 406874902 \xrightarrow{48097086} 454971988$$

Notice that the total time is $454971987 + 48097086 = 503069073$.

Explanation of the third sample: Since all portals are initially off, the ant will not be teleported and will go straight from 0 to

$x_n + 1 = 899754846 + 1 = 899754847.$

G. A Serious Referee

time limit per test: 1 second
memory limit per test: 512 megabytes
input: standard input
output: standard output

Andrea has come up with what he believes to be a novel sorting algorithm for arrays of length n. The algorithm works as follows.

Initially there is an array of n integers $a_1, a_2, \ldots, a_n$. Then, k steps are executed.

For each $1 \le i \le k$, during the i-th step the subsequence of the array a with indexes $j_{i,1} < j_{i,2} < \cdots < j_{i,q_i}$ is sorted, without changing the values with the remaining indexes. So, the subsequence $a_{j_{i,1}}, a_{j_{i,2}}, \ldots, a_{j_{i,q_i}}$ is sorted and all other elements of a are left untouched.

Andrea, being eager to share his discovery with the academic community, sent a short paper describing his algorithm to the journal "Annals of Sorting Algorithms" and you are the referee of the paper (that is, the person who must judge the correctness of the paper). You must decide whether Andrea's algorithm is correct, that is, if it sorts any array a of n integers.

Input

The first line contains two integers n and k ($1 \le n \le 40$, $0 \le k \le 10$) — the length of the arrays handled by Andrea's algorithm and the number of steps of Andrea's algorithm.

Then k lines follow, each describing the subsequence considered in a step of Andrea's algorithm.

The i-th of these lines contains the integer q_i ($1 \le q_i \le n$) followed by q_i integers $j_{i,1}, j_{i,2}, \ldots, j_{i,q_i}$ ($1 \le j_{i,1} < j_{i,2} < \cdots < j_{i,q_i} \le n$) — the length of the subsequence considered in the i-th step and the indexes of the subsequence.

Output

If Andrea's algorithm is correct print ACCEPTED, otherwise print REJECTED.

Examples

input
4 3 3 1 2 3 3 2 3 4 2 1 2
output
ACCEPTED

input
4 3 3 1 2 3 3 2 3 4 3 1 3 4
output
REJECTED

input
3 4 1 1 1 2 1 3 2 1 3
output
REJECTED

input
5 2 3 2 3 4 5 1 2 3 4 5
output
ACCEPTED

Note

Explanation of the first sample: The algorithm consists of 3 steps. The first one sorts the subsequence $[a_1, a_2, a_3]$, the second one sorts the subsequence $[a_2, a_3, a_4]$, the third one sorts the subsequence $[a_1, a_2]$. For example, if initially $a = [6, 5, 6, 3]$, the algorithm transforms the array as follows (the subsequence that gets sorted is highlighted in red)

$$[6, 5, 6, 3] \to [5, 6, 6, 3] \to [5, 3, 6, 6] \to [3, 5, 6, 6]\,.$$

One can prove that, for any initial array a, at the end of the algorithm the array a will be sorted.

Explanation of the second sample: The algorithm consists of 3 steps. The first one sorts the subsequence $[a_1, a_2, a_3]$, the second one sorts the subsequence $[a_2, a_3, a_4]$, the third one sorts the subsequence $[a_1, a_3, a_4]$. For example, if initially $a = [6, 5, 6, 3]$, the algorithm transforms the array as follows (the subsequence that gets sorted is highlighted in red)

$$[\color{red}{6, 5, 6}, 3] \to [5, \color{red}{6, 6, 3}] \to [5, 3, \color{red}{6, 6}] \to [5, 3, 6, 6]\,.$$

Notice that $a = [6, 5, 6, 3]$ is an example of an array that is not sorted by the algorithm.

Explanation of the third sample: The algorithm consists of 4 steps. The first 3 steps do nothing because they sort subsequences of length 1, whereas the fourth step sorts the subsequence $[a_1, a_3]$. For example, if initially $a = [5, 6, 4]$, the algorithm transforms the array as follows (the subsequence that gets sorted is highlighted in red)

$$[\color{red}{5}, 6, \color{red}{4}] \to [\color{red}{5}, 6, \color{red}{4}] \to [\color{red}{5}, 6, \color{red}{4}] \to [\color{red}{5}, 6, \color{red}{4}] \to [4, 6, 5]\,.$$

Notice that $a = [5, 6, 4]$ is an example of an array that is not sorted by the algorithm.

Explanation of the fourth sample: The algorithm consists of 2 steps. The first step sorts the subsequences $[a_2, a_3, a_4]$, the second step sorts the whole array $[a_1, a_2, a_3, a_4, a_5]$. For example, if initially $a = [9, 8, 1, 1, 1]$, the algorithm transforms the array as follows (the subsequence that gets sorted is highlighted in red)

$$[9, \color{red}{8, 1, 1}, 1] \to [\color{red}{9, 1, 1}, \color{red}{8, 1}] \to [1, 1, 1, 8, 9]\,.$$

Since in the last step the whole array is sorted, it is clear that, for any initial array a, at the end of the algorithm the array a will be sorted.

H. Guess the Perimeter

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Let us call a point of the plane *admissible* if its coordinates are positive integers less than or equal to 200.

There is an invisible rectangle such that:

- its vertices are all admissible;
- its sides are parallel to the coordinate axes;
- its area is strictly positive.

Your task is to guess the perimeter of this rectangle.
In order to guess it, you may ask at most 4 queries.

In each query, you choose a nonempty subset of the admissible points and you are told how many of the chosen points are inside or on the boundary of the invisible rectangle.

Interaction

To ask a query (of the kind described in the statement), you shall print two lines:

- In the first line print "`? k`" (without the quotes) where k ($k \geq 1$) is the number of chosen points.
- In the second line print $2k$ integers x_1, y_1, x_2, y_2, ..., x_k, y_k ($1 \leq x_i, y_i \leq 200$ for $i = 1, 2, \ldots, k$) where (x_1, y_1), (x_2, y_2), (x_3, y_3), ..., (x_k, y_k) are the k distinct admissible chosen points (the order of the points is not important).

After this, you should read an integer — the number of chosen points that are inside or on the boundary of the invisible rectangle.
When you have identified the perimeter p of the invisible rectangle, you must print "`! p`" (without quotes) and terminate your program.

If you ask more than 4 queries or if one of the queries is malformed, the interactor terminates immediately and your program receives verdict `Wrong Answer`.

The interactor may be adaptive (i.e., the hidden rectangle may not be chosen before the beginning of the interaction).

After printing a query do not forget to output end of line and flush the output. Otherwise, you will get `Idleness limit exceeded`.
To do this, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `System.out.flush()` in Java;
- `flush(output)` in Pascal;
- `stdout.flush()` in Python;
- see documentation for other languages.

Hacks

To hack a solution, use the following format.

The input has only one line, containing the 4 integers x_0, y_0, x_1, y_1 ($1 \le x_0 < x_1 \le 200$, $1 \le y_0 < y_1 \le 200$) — (x_0, y_0) is the bottom-left vertex of the hidden rectangle and (x_1, y_1) is the top-right vertex of the hidden rectangle.

Note that for hacks the interaction won't be adaptive.

Examples

input
13 5 123 80
output

input
2 2 4 4
output

input
1 1 200 200
output

Note

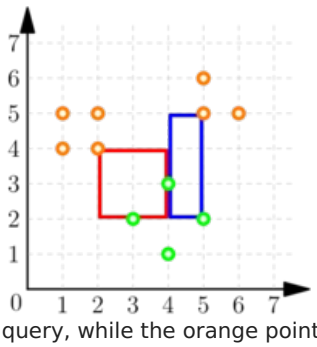
The following is an example of interaction for the **first sample** intended to show the format of the queries.

Query (contestant program)	Answer (interactor)	Explanation
? 4 13 5 13 80 123 5 123 80	4	We choose the 4 vertices of the hidden rectangle.
? 5 100 4 100 81 12 40 124 40 50 50	1	We choose 4 points just outside the hidden rectangle and also the point $(50, 50)$.
? 2 200 200 1 1	0	We choose the points $(1, 1)$ and $(200, 200)$.
! 370		This is the correct perimeter.

For the **second sample**, a possible interaction is the following.

Query (contestant program)	Answer (interactor)	Explanation
? 4 3 2 4 1 5 2 4 3	2	We choose the points $(3, 2)$, $(4, 1)$, $(5, 2)$ and $(4, 3)$.
? 7 1 4 2 4 1 5 2 5 5 5 5 6 6 5	1	We choose the points $(1, 4)$, $(2, 4)$, $(1, 5)$, $(2, 5)$, $(5, 5)$, $(5, 6)$ and $(6, 5)$.
! 8		This is the correct perimeter.

The situation is shown in the following picture:



The green points are the ones belonging to the first query, while the orange points are the ones belonging to the second query. One can see that there are exactly two rectangles consistent with the interactor's answers:

- the rectangle of vertices $(2, 2)$ and $(4, 4)$, shown in red;
- the rectangle of vertices $(4, 2)$ and $(5, 5)$, shown in blue.

Since both of these rectangles have perimeter 8, this is the final answer.

I. Organizing a Music Festival

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are the organizer of the famous "Zurich Music Festival". There will be n singers who will perform at the festival, identified by the integers $1, 2, \ldots, n$. You must choose in which order they are going to perform on stage.

You have m friends and each of them has a set of favourite singers. More precisely, for each $1 \leq i \leq m$, the i-th friend likes singers $s_{i,1}, s_{i,2}, \ldots, s_{i,q_i}$.

A friend of yours is happy if the singers he likes perform consecutively (in an arbitrary order). An ordering of the singers is valid if it makes all your friends happy.

Compute the number of valid orderings modulo $998\,244\,353$.

Input

The first line contains two integers n and m ($1 \leq n,\ m \leq 100$) — the number of singers and the number of friends correspondingly.

The i-th of the next m lines contains the integer q_i ($1 \leq q_i \leq n$) — the number of favorite singers of the i-th friend – followed by the q_i integers $s_{i,1}, s_{i,2}, \ldots, s_{i,q_i}$ ($1 \leq s_{i,1} < s_{i,2} < \cdots < s_{i,q_i} \leq n$) — the indexes of his favorite singers.

Output

Print the number of valid orderings of the singers modulo $998\,244\,353$.

Examples

input
3 1 2 1 3
output
4

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

input
100 1 1 50
output
35305197

input
5 1 5 1 2 3 4 5
output
120

input
2 5 1 2 1 2 1 2 1 1 1 1
output
2

input
11 4 5 4 5 7 9 11 2 2 10 2 9 11 7 1 2 3 5 8 10 11
output
384

Note

Explanation of the first sample: There are 3 singers and only 1 friend. The friend likes the two singers 1 and 3. Thus, the 4 valid orderings are:

- 1 3 2
- 2 1 3
- 2 3 1
- 3 1 2

Explanation of the second sample: There are 5 singers and 5 friends. One can show that no ordering is valid.

Explanation of the third sample: There are 100 singers and only 1 friend. The friend likes only singer 50, hence all the $100!$ possible orderings are valid.

Explanation of the fourth sample: There are 5 singers and only 1 friend. The friend likes all the singers, hence all the $5! = 120$ possible orderings are valid.
