Problem A. Mad Max

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

You are given a permutation $a_1, a_2, \ldots, a_n$ of the integers 1 through n. That is, each integer from 1 to n occurs exactly once in a. You are also given an array $b_1, b_2, \ldots, b_n$ $(1 \le b_i \le n)$, whose values may repeat.

Consider a permutation $p_1, p_2, \ldots, p_n$ of the integers 1 through n, where each p_i is read as a position in a. We call p *valid* if, for every index i, the position p_i lies inside some interval of a whose maximum value equals b_i. Formally, for each i $(1 \le i \le n)$ there must exist an interval $[l_i, r_i]$ $(1 \le l_i \le r_i \le n)$ with $l_i \le p_i \le r_i$ and $\max(a_{l_i}, a_{l_i+1}, \ldots, a_{r_i}) = b_i$.

Count the number of valid permutations p. Since the answer may be large, output it modulo 998244353.

Input

The first line contains a single integer t $(1 \le t \le 10^4)$ — the number of test cases.

The first line of each test case contains a single integer n $(1 \le n \le 3 \cdot 10^5)$ — the length of the permutation.

The second line contains n integers $a_1, a_2, \ldots, a_n$ — a permutation of the integers 1 through n.

The third line contains n integers $b_1, b_2, \ldots, b_n$ $(1 \le b_i \le n)$ — the elements of the array b.

It is guaranteed that the sum of n over all test cases does not exceed $3 \cdot 10^5$.

Output

For each test case, print a single integer — the number of valid permutations p, taken modulo 998244353.

Example

standard input	standard output
4 3 2 3 1 3 2 1 3 1 3 2 1 1 2 5 3 1 2 5 4 5 5 3 5 4 3 1 2 3 3 3 3	1 0 18 6

Note

In the first test case, $a = [2, 3, 1]$ and $b = [3, 2, 1]$. Exactly one permutation is valid: $p = [2, 1, 3]$. To check it, for index 1 take the interval $[1, 2]$, which contains $p_1 = 2$ and has maximum $\max(a_1, a_2) = 3 = b_1$. For index 2 take $[1, 1]$, which contains $p_2 = 1$ and has maximum $2 = b_2$. And for index 3 take $[3, 3]$, which contains $p_3 = 3$ and has maximum $1 = b_3$.

In the second test case, it can be shown that no valid permutation exists. So the answer is 0.

Problem B. Difference Engine

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

You are given a positive integer n. Count the number of pairs of integers (a, b) with $1 \le a < b \le n$ such that $(b - a)$ divides $a \cdot b$.

A non-zero integer x divides an integer y if there exists an integer z such that $y = x \cdot z$.

Input

The first line contains a single integer t ($1 \le t \le 10^6$) — the number of test cases.

The first and only line of each test case contains a single integer n ($2 \le n \le 2 \cdot 10^6$).

Output

For each test case, output a single integer — the number of pairs (a, b) that satisfy the condition.

Example

standard input	standard output
3 2 5 10	1 5 19

Note

In the first test case, $n = 2$, and the only pair is $(1, 2)$: here $b - a = 1$ divides $a \cdot b = 2$, so the answer is 1.

In the second test case, $n = 5$, the pairs that satisfy the condition are:

- $(1, 2)$, since 1 divides 2;
- $(2, 3)$, since 1 divides 6;
- $(2, 4)$, since 2 divides 8;
- $(3, 4)$, since 1 divides 12;
- $(4, 5)$, since 1 divides 20.

For example, the pair $(2, 5)$ does not count, because $b - a = 3$ does not divide $a \cdot b = 10$. In total there are 5 valid pairs.

Problem C. Parallel Lives

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

You are given $2n$ distinct points on a two-dimensional plane. You must partition these $2n$ points into exactly n pairs (that is, every point must belong to exactly one pair) and connect each pair of points with a straight line segment, producing n line segments in total.

Find a pairing such that all of the n segments are parallel to one another and no two of them intersect, meaning no two segments share a common point. If such a pairing exists, print any one of them; otherwise, report that it is impossible.

Input

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

The first line of each test case contains a single integer n ($1 \le n \le 1000$) — the number of pairs.

Each of the next $2n$ lines contains two integers x_i and y_i ($-10^9 \le x_i, y_i \le 10^9$) — the coordinates of the i-th point.

It is guaranteed that all $2n$ points in each test case are distinct and that the sum of n over all test cases does not exceed 1000.

Output

For each test case, if no valid pairing exists, print -1.

Otherwise, print n lines. Each line should contain two integers u_i and v_i ($1 \le u_i, v_i \le 2n$, $u_i \ne v_i$) — denoting that the u_i-th and v_i-th points form a pair.

If there are multiple valid pairings, output any of them.

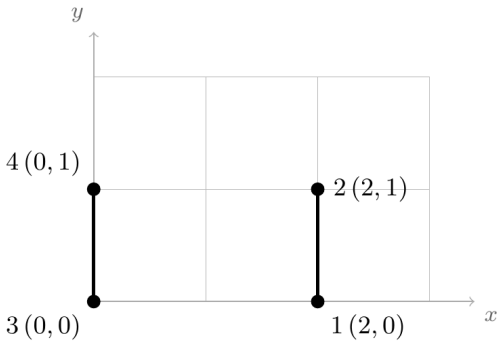
Example

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

Note

In the first test case, pairing points 1 and 2 and pairing points 3 and 4 produces two vertical segments.

They are parallel and do not touch, so this pairing is valid.



In the second test case, each of the three possible pairings produces two segments that are not parallel, so no valid pairing exists and the answer is -1.

Problem D. Double Blind

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

This is an interactive problem.

There are two hidden integers a and b satisfying $0 \le a < b < 2^{60}$.

You may ask queries to learn about the two hidden integers. In one query you choose two integers m and v ($0 \le m, v < 2^{60}$). A hidden integer x *satisfies* the query if $(x \,\&\, m) = v$, where & denotes the bitwise AND operation. The response is the number of the two hidden integers that satisfy the query, taken modulo 2. That is, if neither or both of a and b satisfy the query, the response is 0; otherwise, exactly one of a and b satisfies the query, and the response is 1.

Your task is to determine both of the hidden integers a and b using at most 125 queries.

Interaction Protocol

The first line of the input contains a single integer t ($1 \le t \le 1000$) — the number of test cases. For each test case, the interaction proceeds as follows. Each test case has its own hidden pair (a, b).

To make a query, print one line of the form `?` m v ($0 \le m, v < 2^{60}$). After each query, read a single integer r — the response defined above (either 0 or 1).

When you have determined the hidden integers, print one line of the form `!` a b, the two hidden integers in increasing order. Printing the answer does not count as a query. After that, either proceed to the next test case or terminate your program if this was the last one.

You can make at most 125 queries per test case; exceeding this limit, or printing an invalid query, results in the `Wrong Answer` verdict.

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

- `fflush(stdout)` in C;
- `fflush(stdout)` or `std::cout.flush()` in C++;
- `System.out.flush()` in Java;
- `sys.stdout.flush()` in Python;
- `std::io::stdout().flush()` in Rust;
- Appropriate `flush` operation in any language.

Example

standard input	standard output
1 0 1 0	 ? 7 5 ? 11 9 ? 6 3 ! 5 13

Note

In the sample there is one test case, with hidden integers $a = 5$ and $b = 13$. The queries below only illustrate the format — they are not enough to determine the answer in general.

Solution	Jury	Explanation
	`1`	There is one test case.
`? 7 5`	`0`	Here $5 \,\&\, 7 = 5$ and $13 \,\&\, 7 = 5$, both equal to $v = 5$, so both of the hidden integers satisfy the condition and the response is 0.
`? 11 9`	`1`	Here $13 \,\&\, 11 = 9$ equals $v = 9$, but $5 \,\&\, 11 = 1$ does not, so exactly one hidden integer satisfies the condition and the response is 1.
`? 6 3`	`0`	Here $5 \,\&\, 6 = 4$ and $13 \,\&\, 6 = 4$. Neither value equals 3, so the response is 0.
`! 5 13`		The solution reports the hidden integers $a = 5$ and $b = 13$.

Empty lines in the example input and output are given only for better readability; you don't need to output them in your solution.

Problem E. A Distinct Problem

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

You are given an array a of n positive integers. In one operation, you may delete any single element of the array.

Call a non-empty array *good* if its largest element minus its smallest element is strictly less than the number of distinct values it contains. Formally, a non-empty array b with k distinct values is good if $\max(b) - \min(b) < k$, where $\max(b)$ and $\min(b)$ are the largest and smallest elements of b, respectively.

For example, $[7, 7]$ is good, since $\max(b) - \min(b) = 0$ is less than its 1 distinct value. But the array $[2, 9]$ is not good, since $\max(b) - \min(b) = 7$, but it has only 2 distinct values, and $7 \not< 2$.

Determine the minimum number of operations needed to make the array a good. It is always possible to do so, since any array consisting of a single element is good.

Input

The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases.

The first line of each test case contains a single integer n ($1 \le n \le 3 \cdot 10^5$) — the length of the array.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — the elements of the array.

It is guaranteed that the sum of n over all test cases does not exceed $3 \cdot 10^5$.

Output

For each test case, output a single integer — the minimum number of operations needed to make the array good.

Example

standard input	standard output
2 4 8 6 2 4 2 7 9	3 1

Note

In the first test case, deleting 8, 6, and 4 leaves [2], which is good ($\max - \min = 0$, less than its 1 distinct value). No sequence of fewer than 3 deletions works.

Problem F. Shortest Paths

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

You are given a directed graph with n vertices and m edges that is weakly connected*. The vertices are numbered from 1 to n, and each edge is colored either black or white.

Your task is to assign a positive integer weight w_i ($1 \le w_i \le 10^9$) to each edge i ($1 \le i \le m$) such that:

- every black edge lies on at least one shortest path† from vertex 1 to vertex n, and
- no white edge lies on any shortest path from vertex 1 to vertex n.

It is guaranteed that at least one path from vertex 1 to vertex n exists. If no valid assignment exists, output -1; otherwise, output the assigned weights. If there are multiple valid assignments, you may output any of them.

*A directed graph is weakly connected if, after ignoring the directions of the edges, there is a path between every pair of vertices.

†A path from 1 to n is a sequence of directed edges in which the first edge starts at vertex 1, the last edge ends at vertex n, and each edge after the first starts where the previous one ended. Its length is the sum of the weights of its edges, and a shortest path is one of minimum length.

Input

The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases.

The first line of each test case contains two integers n and m ($2 \le n \le 3 \cdot 10^5$, $1 \le m \le \min(n(n-1), 3 \cdot 10^5)$) — the number of vertices and the number of edges.

Each of the next m lines contains three integers u_i, v_i, and c_i ($1 \le u_i, v_i \le n$; $u_i \ne v_i$; $c_i \in \{0, 1\}$) — describing a directed edge from vertex u_i to vertex v_i. If $c_i = 1$, the edge is black; if $c_i = 0$, the edge is white. It is guaranteed that there is at most one edge for each ordered pair of vertices.

It is guaranteed that the sum of n over all test cases does not exceed $3 \cdot 10^5$, and the sum of m over all test cases does not exceed $3 \cdot 10^5$.

Output

For each test case, if there is no valid assignment of weights, output -1. Otherwise, output m integers $w_1, w_2, \ldots, w_m$ ($1 \le w_i \le 10^9$) — the assigned weight for each edge, in the order the edges appear in the input. If there are multiple valid assignments, you may output any of them.

Example

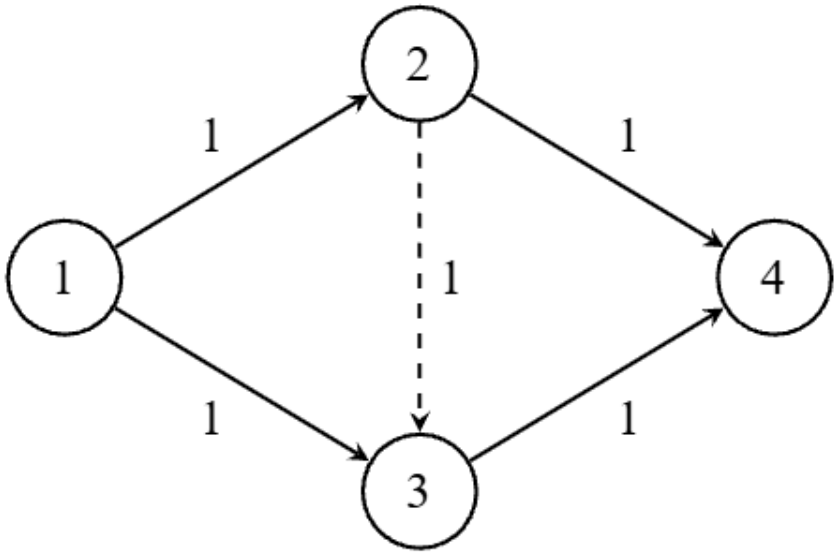
standard input	standard output
3 3 3 1 2 1 2 3 1 1 3 0 3 2 1 2 1 2 3 0 4 5 1 2 1 2 4 1 1 3 1 3 4 1 2 3 0	1 2 4 -1 1 1 1 1 1

Note

In the first test case, assign weight 1 to edge $(1 \to 2)$, weight 2 to edge $(2 \to 3)$, and weight 4 to edge $(1 \to 3)$ (so the output is 1 2 4). Then the path $1 \to 2 \to 3$ has total weight 3. Since $3 < 4$, the white edge $(1 \to 3)$ is not on a shortest path, while the black edges $(1 \to 2)$ and $(2 \to 3)$ lie on the shortest path.

In the second test case, $(1 \to 2)$ is black. To reach 3, we must use the edge $(2 \to 3)$, which is white, so every path from 1 to 3 — and in particular every shortest path — includes this white edge. Thus no valid assignment exists.

In the third test case, by assigning weight 1 to every edge, the shortest path distance from 1 to 4 is 2 (via $1 \to 2 \to 4$ or $1 \to 3 \to 4$). The white edge $(2 \to 3)$ is not on any shortest path, since any path using it has length at least 3.



Problem G. Fairy Lights

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

You are given a tree on n vertices. Every vertex carries a light, and initially all n lights are on.

You then repeatedly perform the following operation:

- choose an edge whose two endpoints are both currently on, and turn off the lights at both of them.

Once a light is turned off it never turns back on. You keep performing the operation until no edge has both of its endpoints on, that is, until the operation can no longer be performed.

Depending on the order in which you choose the edges, you may end up with different sets of vertices whose lights are still on. Count the number of distinct such sets of vertices that are reachable when you stop. Since this number can be large, output it modulo 998244353.

Input

The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases.

The first line of each test case contains a single integer n ($1 \le n \le 3 \cdot 10^5$) — the number of vertices of the tree.

Each of the next $n-1$ lines contains two integers u and v ($1 \le u, v \le n$; $u \ne v$) — an edge of the tree. It is guaranteed that the given edges form a tree.

It is guaranteed that the sum of n over all test cases does not exceed $3 \cdot 10^5$.

Output

For each test case, output a single integer — the number of distinct reachable sets of on lights when you stop, modulo 998244353.

Example

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

Note

In the first test case, the tree is the path $1 - 2 - 3$. There are two ways the process can go:

- Turn off the lights at the endpoints of edge $1 - 2$. Now only vertex 3 is on, and no edge has both endpoints on, so you stop. The set of on lights is $\{3\}$.
- Turn off the lights at the endpoints of edge $2 - 3$. Now only vertex 1 is on, and you stop. The set of on lights is $\{1\}$.

These give two distinct final sets, $\{3\}$ and $\{1\}$, so the answer is 2.

In the second test case, the tree is a star with center 1. Whichever edge $1 - i$ you choose first, vertices 1 and i turn off and the two remaining leaves stay on with no edge left to use. The reachable final sets are $\{3, 4\}$, $\{2, 4\}$, and $\{2, 3\}$, so the answer is 3.

In the third test case, the tree is the path $1 - 2 - 3 - 4$. If you first use edge $2 - 3$, you stop with vertices 1 and 4 on. Otherwise you first use one of the end edges, after which the opposite end edge still has both endpoints on and must be used, leaving every light off. The reachable final sets are $\{1, 4\}$ and $\{\,\}$, so the answer is 2.

Problem H. Standoff

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

You are given a string s consisting of lowercase English letters.

Find two non-empty substrings* p and q of s such that p is not a substring of q, and q is not a substring of p. Note that p and q do not have to be disjoint — they may correspond to overlapping positions in s. Over all such pairs (p, q), output the maximum possible value of $|p| + |q|$, where $|x|$ denotes the length of a string x.

If no such pair exists, output 0.

*A string a is a substring of a string b if a can be obtained from b by deletion of several (possibly, zero or all) characters from the beginning and several (possibly, zero or all) characters from the end. For example, `us` is a substring of `sust`, but `ut` and `aus` are not.

Input

The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases.

The first and only line of each test case contains a string s ($1 \le |s| \le 3 \cdot 10^5$) consisting of lowercase English letters.

It is guaranteed that the sum of $|s|$ over all test cases does not exceed $3 \cdot 10^5$.

Output

For each test case, output a single integer — the maximum value of $|p| + |q|$ over all valid pairs, or 0 if no valid pair exists.

Example

standard input	standard output
2 ab z	2 0

Note

In the first test case, $s =$ `ab`. Choose $p =$ `a` and $q =$ `b`; neither string is a substring of the other, and $|p| + |q| = 1 + 1 = 2$, which is the maximum possible.

In the second test case, $s =$ `z` has only one substring, `z` itself, so the only possible choice is $p = q =$ `z`. But then p is a substring of q, so this pair is not valid. No valid pair exists, and the answer is 0.

Problem I. Spot the Difference

Input file:	`standard input`
Output file:	`standard output`
Time limit:	3 seconds
Memory limit:	512 megabytes

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

Call a sequence of integers $b_1, b_2, \ldots, b_n$ *valid* if it satisfies both of the following:

- $0 \le b_i < 2^k$ for every $1 \le i \le n$;
- for all $1 \le i \le j \le n$, the bitwise XOR of the contiguous subarray $a_i, a_{i+1}, \ldots, a_j$ is different from the bitwise XOR of the corresponding contiguous subarray $b_i, b_{i+1}, \ldots, b_j$:

$$a_i \oplus a_{i+1} \oplus \cdots \oplus a_j \neq b_i \oplus b_{i+1} \oplus \cdots \oplus b_j.$$

Here $\oplus$ denotes the bitwise XOR operation.

You have to output two things: the number of valid sequences b, modulo 998244353, and the lexicographically smallest* valid sequence b (if at least one valid sequence exists).

*A sequence b is lexicographically smaller than a sequence c of the same length if, at the first position where they differ, b has a smaller element than the corresponding element in c.

Input

The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases.

The first line of each test case contains two integers n and k ($1 \le n \le 3 \cdot 10^5$, $0 \le k \le 60$) — the length of the sequence and the bit width.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i < 2^k$) — the elements of the sequence.

It is guaranteed that the sum of n over all test cases does not exceed $3 \cdot 10^5$.

Output

For each test case, print the number of valid sequences b modulo 998244353 on the first line.

If at least one valid sequence exists, print on the next line n space-separated integers $b_1, b_2, \ldots, b_n$ — the lexicographically smallest valid sequence. Otherwise, print only the first line for that test case.

Example

standard input	standard output
3 2 2 1 2 2 1 0 1 5 3 0 1 6 4 3	6 0 0 0 2520 1 2 0 3 5

Note

In the first test case, $a = [1, 2]$ and $k = 2$, so each b_i is one of $0, 1, 2, 3$. The three subarray conditions are $b_1 \neq 1$, $b_2 \neq 2$, and $b_1 \oplus b_2 \neq 1 \oplus 2 = 3$. There are 6 sequences that satisfy all three, and $b = [0, 0]$ is the smallest of them: it works because $0 \neq 1$, $0 \neq 2$, and $0 \oplus 0 = 0 \neq 3$.

In the second test case, no valid sequence exists, so the answer is 0 and nothing else is printed.

Problem J. Prefix Split

Input file:	`standard input`
Output file:	`standard output`
Time limit:	3 seconds
Memory limit:	512 megabytes

You are given two strings a and b, each of length n and consisting of lowercase English letters, together with an array of n positive integers $c_1, c_2, \ldots, c_n$.

For a string s, define its *score* $f(s)$ as follows. Consider every way to partition s into a non-empty prefix and a non-empty suffix such that both parts are prefixes of a. Each such split contributes c_j, where j is the length of the suffix, and $f(s)$ is the sum of these contributions. In particular, $f(s) = 0$ if there is no such split. Formally, let m be the length of s; then

$$f(s) = \sum_{\substack{1 \le i < m \\ s[1..i] \text{ and } s[i+1..m] \text{ are prefixes of } a}} c_{m-i}.$$

You must answer q queries. Each query gives you two integers l and r $(1 \le l \le r \le n)$. Output $f(b[l..r])$, where $b[l..r] = b_l b_{l+1} \ldots b_r$ is the substring of b starting at index l and ending at index r.

Input

The first line contains a single integer t $(1 \le t \le 10^4)$ — the number of test cases.

The first line of each test case contains two integers n and q $(1 \le n, q \le 10^6)$ — the length of the strings and the number of queries.

The second line contains the string a, and the third line contains the string b; each has length n and consists of lowercase English letters.

The fourth line contains n integers $c_1, c_2, \ldots, c_n$ $(1 \le c_i \le 10^9)$ — the array c.

Each of the next q lines contains two integers l and r $(1 \le l \le r \le n)$ — describing one query.

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

Output

For each query, print a single integer — the score $f(b[l..r])$.

Example

standard input	standard output
2 5 5 abaab ababa 2 3 4 5 6 1 5 1 4 3 5 2 4 1 1 4 3 aaaa aaaa 1 10 100 1000 1 4 2 4 1 2	4 3 2 0 0 111 11 1

Note

In the first test case, $a = \texttt{abaab}$ and $b = \texttt{ababa}$.

For the first query, $b[1..5] = \texttt{ababa}$. There are four ways to split it into two non-empty parts:

- a and baba — baba is not a prefix of a;
- ab and aba — both are prefixes of a, so this split contributes c_3, as the suffix aba has length 3;
- aba and ba — ba is not a prefix of a;
- abab and a — abab is not a prefix of a.

Only the second split contributes, so $f(\texttt{ababa}) = c_3 = 4$.

For the fourth query, $b[2..4] = \texttt{bab}$. No prefix of a starts with b, so no split contributes and $f(\texttt{bab}) = 0$.

In the second test case, $a = b = \texttt{aaaa}$. For the first query, $b[1..4] = \texttt{aaaa}$: all three splits contribute, so $f(\texttt{aaaa}) = c_1 + c_2 + c_3 = 1 + 10 + 100 = 111$.

Problem K. Prime Coloring

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

You are given a positive integer n and k distinct prime numbers $p_1, p_2, \ldots, p_k$.

You color each of the integers $1, 2, \ldots, n$ with one of two colors, 0 or 1. Such a coloring is *good* if, for every index i $(1 \le i \le k)$ and every positive integer x with $p_i \cdot x \le n$, the integers x and $p_i \cdot x$ receive different colors.

In addition, the colors of some integers are fixed in advance. You are given m pairs $(v_1, c_1), (v_2, c_2), \ldots, (v_m, c_m)$. A coloring is *valid* if it is good and, for every index j $(1 \le j \le m)$, the integer v_j has color c_j.

Count the number of valid colorings. Since the answer can be large, output it modulo 998244353.

Input

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

The first line of each test case contains two integers n and k $(1 \le n \le 10^{18}, 1 \le k \le 16)$ — the number of integers to color and the number of primes.

The second line contains k distinct prime numbers $p_1, p_2, \ldots, p_k$ $(2 \le p_i \le 10^9)$ — the given primes.

The third line contains a single integer m $(0 \le m \le 3 \cdot 10^5)$ — the number of integers whose color is fixed.

The j-th of the next m lines contains two integers v_j and c_j $(1 \le v_j \le n, c_j \in \{0, 1\})$ — the integer v_j is fixed to the color c_j. The values $v_1, v_2, \ldots, v_m$ are distinct.

It is guaranteed that the sum of m over all test cases does not exceed $3 \cdot 10^5$.

Output

For each test case, output a single integer — the number of valid colorings, modulo 998244353.

Example

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

Note

In the first test case, $n = 4$, the only prime is 2, and the color of 1 is fixed to 0. There are exactly two valid colorings; giving the colors of $1, 2, 3, 4$ in order, they are $(0, 1, 0, 0)$ and $(0, 1, 1, 0)$. In each, 1 and 2 have different colors, 2 and 4 have different colors, and 1 has color 0.

In the second test case, $n = 6$ and the primes are 2 and 3. The colors of 2 and 6 are both fixed to 0, but $6 = 3 \cdot 2 \le 6$, so a good coloring must give them different colors — a contradiction. Hence no valid coloring exists and the answer is 0.

Problem L. Appending LIS

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

For an integer array b and a given integer k, define $S(b)$ as follows. Perform exactly k operations; in each operation, compute the length of the longest strictly increasing subsequence* (LIS) of the current array b and append this value to the end of b. Each operation acts on the array obtained from the previous ones. After the k operations, $S(b)$ is the sum of all elements of the resulting array.

You are given an integer array a of length n and the integer k. For each i from 1 to n, compute $S(a[1..i])$, where $a[1..i]$ denotes the prefix $a_1, a_2, \ldots, a_i$. Each prefix is processed independently: the operations performed while computing $S(a[1..i])$ do not affect any other prefix.

*A subsequence is obtained from an array by deleting zero or more elements without changing the order of the remaining ones; it is strictly increasing if each of its elements is strictly greater than the previous one.

Input

The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases.

The first line of each test case contains two integers n and k ($1 \le n \le 3 \cdot 10^5$, $1 \le k \le 10^9$) — the length of the array and the number of operations.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($-10^9 \le a_i \le 10^9$) — the elements of the array.

It is guaranteed that the sum of n over all test cases does not exceed $3 \cdot 10^5$.

Output

For each test case, output n integers on a single line — the values $S(a[1..1]), S(a[1..2]), \ldots, S(a[1..n])$.

Example

standard input	standard output
3 3 2 1 3 0 3 10 1 1 1 5 1 2 1 -1 3 2	3 8 8 11 12 13 3 4 3 7 9

Note

In the first test case, $a = [1, 3, 0]$ and $k = 2$. Each prefix is processed independently.

- Prefix $[1]$: the LIS has length 1, so we append 1, giving $[1, 1]$. The LIS is still 1 (the two equal values 1 cannot both belong to a strictly increasing subsequence), so we append 1 again, giving $[1, 1, 1]$. The sum is $1 + 1 + 1 = 3$.
- Prefix $[1, 3]$: the LIS has length 2, so we append 2, giving $[1, 3, 2]$. The LIS is still 2, so we append 2 again, giving $[1, 3, 2, 2]$. The sum is $1 + 3 + 2 + 2 = 8$.
- Prefix $[1, 3, 0]$: the LIS has length 2, so we append 2, giving $[1, 3, 0, 2]$. The LIS is still 2, so we append 2 again, giving $[1, 3, 0, 2, 2]$. The sum is $1 + 3 + 0 + 2 + 2 = 8$.

So the answer for the first test case is 3 8 8.