Bay Area Programming Contest 2025

BAPC 2025

March 2, 2025

Problems

Do not open before the contest has started.

Reference Sheet

If you are unfamiliar with some of the terminology in this contest, you can check this sheet.

Bitwise Operations

The *binary representation* of a non-negative integer *n* is the result of writing it out in base 2. For example:

- The binary representation of 13 is 1101, since $13 = 2^3 + 2^2 + 2^0$.
- The binary representation of 67 is 1000011, since $67 = 2^6 + 2^1 + 2^0$.
- The binary representation of 31 is 11111, since $31 = 2^4 + 2^3 + 2^2 + 2^1 + 2^0$.
- The binary representation of 0 is 0.

Since any non-negative integer can be written uniquely as a sum of powers of 2, this binary representation is unique.

For a non-negative integer *i*, we say that the i-th bit in the binary representation of *n* is *set* if and only if there is a 1 at the corresponding position. For example:

- The 0-th, 2-nd, and 3-rd bits of 13 are set, but the 1-st and 4-th are not.
- The 0-th, 1-st, and 6-th bits of 67 are set, but the 2-nd, 3-rd, 4-th, 5-th, and 7-th are not.
- The 1000-th bit of 67 is not set.

The *bitwise and* of two positive integers *n* and *m*, denoted `n&m`, is computed as follows. The i-th bit in `n&m` will be set if and only if *n* and *m* both have the i-th bit set. For example:

- The *bitwise and* of 13 and 31 is 1101, or 13 in decimal.
- The *bitwise and* of 67 and 31 is 11, or 3 in decimal.
- The *bitwise and* of 0 and 31 is 0.

Similarly, the *bitwise or* of *n* and *m*, written as `n|m`, has its i-th bit set if either *n* or *m* has its i-th bit set. For example:

- The *bitwise or* of 13 and 31 is 11111, or 31 in decimal.
- The *bitwise or* of 67 and 31 is 1011111, or 95 in decimal.
- The *bitwise or* of 0 and 31 is 11111, or 31 in decimal.

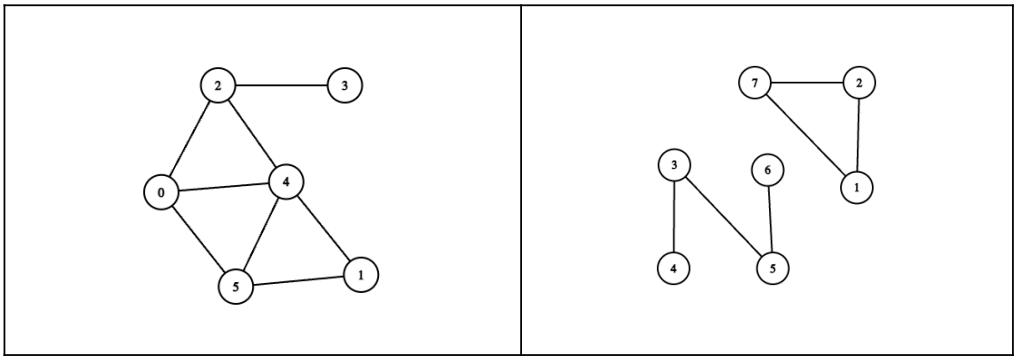
Finally, the *bitwise XOR* of *n* and *m*, written `n^m`, has its i-th bit set if exactly one of *n* and *m* has its i-th bit set. For example:

- The *bitwise XOR* of 13 and 31 is 10010, or 18 in decimal.
- The *bitwise XOR* of 67 and 31 is 1011100, or 92 in decimal.
- The *bitwise XOR* of 0 and 31 is 11111, or 31 in decimal.

In most programming languages, you can compute these operations using the same notation: `n&m`, `n|m`, and `n^m` for AND, OR, XOR respectively.

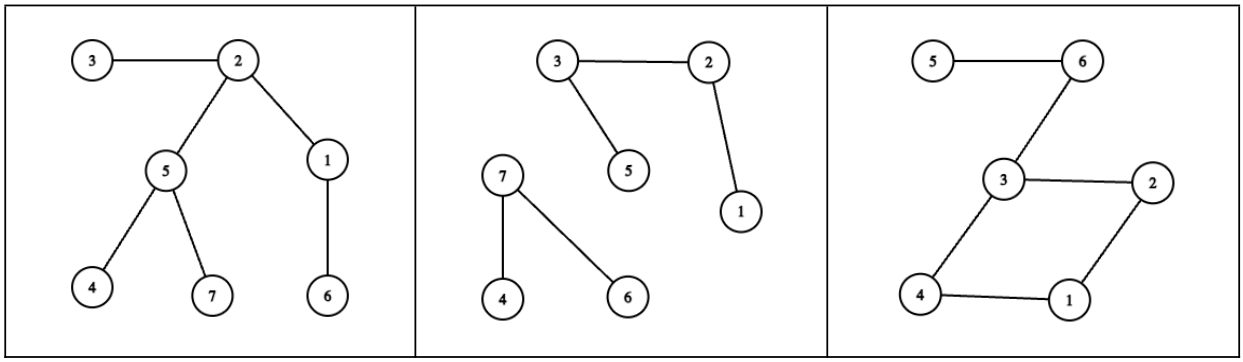
Graphs

A *graph* is a set of nodes, along with a set of edges. Informally, you can think of nodes as being “cities” and edges as being “roads” between the cities.

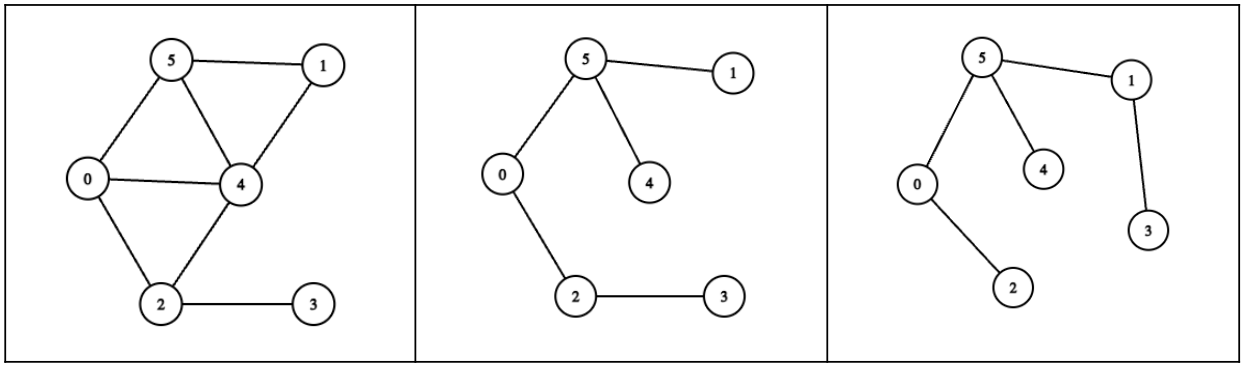


We say a graph is *connected* if it is possible to reach any node from any other node by traversing edges. For example, the graph to the left above is *connected*, but the graph on the right is not, because, for example, you cannot reach node 7 from node 6.

A *tree* is a connected graph, such that there is exactly one simple path between any two vertices. A *simple path* is a path that visits each vertex at most once. For example, the graph on the left below is a *tree*, but the one in the middle is not (it is not connected) and neither is the one on the right (it has multiple paths between vertices 1 and 3).



A *spanning tree* of a connected graph is a subset of its edges that form a tree. For example, the graph in the middle below is a spanning tree of the graph on the left, but the one on the right is not (it contains an edge that was not in the original graph).



Problem A. A Times B

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	1024 megabytes

Boxlunch is on a map which can be modeled as an infinite grid of cells, with cell (i, j) having value $i \times j$. Boxlunch starts at cell (a, b) and is trying to reach cell (n, m), where $n \geq a$ and $m \geq b$.

If Boxlunch is at cell (x, y), he can either move to cell $(x + 1, y)$ or $(x, y + 1)$.

The total value of his path is the sum of the values of all cells he visits, including his starting and ending cells.

Find the maximum total value of a path he can take to reach cell (n, m).

Input

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

The only line of each test contains four integers a, b, n, and m — Boxlunch's starting location and his target location ($0 \leq a, b \leq 10^6, a \leq n \leq 10^6, b \leq m \leq 10^6$).

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

Output

For each test case, output a single number — the maximum total value over all paths Boxlunch can take to reach cell (n, m).

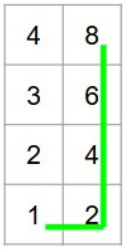
Example

standard input	standard output
6 1 1 3 1 1 1 2 4 1 1 1 1 1 3 4 3 1 4 3 4 3 900000 900000 900000	6 21 1 30 24 364500404997300000

Note

In the first test case, the only path Boxlunch can take is moving to cell $(2, 1)$, then to cell $(3, 1)$, giving a value of $1 \times 1 + 2 \times 1 + 3 \times 1 = 1 + 2 + 3 = 6$.

In the second test case, the following image shows the optimal path Boxlunch could take, giving a value of $1 + 2 + 4 + 6 + 8 = 21$.



In the third test case, Boxlunch doesn't have to move, giving a value of $1 \times 1 = 1$.

In the last test, note that the answer may not fit in a 32-bit integer.

Problem B. Bitstring Accruing Policy Convergence

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

For a bitstring s (meaning s consists of `0`s and `1`s), we define $f(s)$ as the string obtained by replacing each `0` in s with `001`, and each `1` with `01`. For example, $f(\texttt{001}) = \texttt{00100101}$, and $f(\texttt{111}) = \texttt{010101}$.

We write $f^x(s)$ to denote applying f to some string x times. For example, $f^5(s)$ means $f(f(f(f(f(s)))))$.

You are given a list of n distinct integers $a_1, a_2, \ldots, a_n$. You need to find bitstrings s and t that satisfy all the following conditions:

- s and t have the same length.
- The length of s and t is between 1 and $3 \cdot 10^5$ inclusive.
- s and t contain the same number of `0`s, and the same number of `1`s.
- Let $S = f^{22}(s)$ and $T = f^{22}(t)$. Then, a is exactly the list of indices where S and T are different. Or more formally: for all $1 \leq j \leq |S|$, we have $S_j \neq T_j$ if and only if j is an element of a.

Note that this last condition implies that all values a_i must be less than or equal to the length of S and T.

We guarantee that a solution exists in all tests. If there are multiple solutions, you can output any of them.

Input

The first line contains n, the number of positions ($2 \leq n \leq 3 \cdot 10^5$).

The next n lines contain integers $a_1, a_2, \ldots, a_n$, the positions ($1 \leq a_i \leq 10^{18}$).

it is guaranteed that all positions are distinct, and that a solution exists.

Output

Output s and t satisfying the conditions above, on two separate lines.

If there are multiple solutions, you can output any of them.

Examples

standard input	standard output
2 1134903170 1134903171	10 01
4 1134903170 2269806340 1134903171 2269806341	110 011
6 1134903170 1134903171 2269806340 2269806341 4106118243 4106118244	1100001 0101001
8 12318354730 16424472972 2971215074 10482042826 2971215073 10482042827 16424472973 12318354729	0101111100110 0011111001101

Note

Why the magic number 22 in the problem statement? Maybe one of the organizers is a Taylor Swift fan (guess who).

In the first test, s and t differ by exactly two positions. $f(s) = \texttt{01001}$ and $f(t) = \texttt{00101}$, which also differ by exactly two positions. In fact, we can show that for all positive integers n, $f^n(s)$ and $f^n(t)$ differ by exactly two positions.

In the second test, note that solutions such as $s = \texttt{011}$ and $t = \texttt{110}$ would also be accepted.

In the third test, note that solutions such as $s = \texttt{1100}$ and $t = \texttt{0101}$ would also be accepted.

Problem C. Count Pairs

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	1024 megabytes

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

A pair of integers x, y is *good* if $\max(x, y) < x \oplus y$, where $\oplus$ denotes the bitwise XOR operation.

A set S of integers *counts* if all its elements are unique, and all pairs of distinct integers in S are good.

What is the largest subset of $a_1, a_2, \ldots, a_n$ that counts?

Input

The first line contains n ($1 \le n \le 3 \cdot 10^5$).

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 4 \cdot 10^6$).

Output

Print the size of the largest subset of $a_1, a_2, \ldots, a_n$ that counts.

Examples

standard input	standard output
3 1 2 3	2
3 4000000 4000000 4000000	1

Note

In the first test, the set $\{1, 2\}$ counts, since $1 \oplus 2 = 3$, while $\max(1, 2) = 2$, and $3 > 2$. However, $\{1, 2, 3\}$ does not count, since, for example, $1 \oplus 3 = 2 \le \max(1, 3) = 3$. So the answer is 2.

In the second test, the optimal solution is to just take the set $\{4\,000\,000\}$.

This page is intentionally left blank.

Problem D. Drowsy Robots

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

At X-Camp Academy, students don't just learn to code. In fact, coding is merely 5% of what X-Camp helps students to begin with. Instead, they spend most of their time learning to tackle challenges through algorithms and problem-solving. Many of the methodologies are used in the state-of-the-art research and development of artificial intelligence and other technology or science.

Today's task is straight out of a robotics competition: A fleet of n autonomous delivery robots are standing in a line, where the i-th robot is currently at position $x = i$.

But a software bug means they will all be moving at the same speed! At time $t = 0$ seconds, all robots will start moving to the left (in the negative x direction) at a constant speed of 1 unit per second. If they keep going at this rate, they'll get to their destinations at the wrong time and disrupt the entire system. Fortunately, you have a specialized slow-motion gun that can selectively reduce their speed.

When you fire your slow-motion gun, all robots at positions $x \geq 0$ have their current speeds halved. **You are only allowed to fire at positive integer times, i.e. when t is an integer and $t > 0$.** You are allowed to fire multiple times simultaneously.

You are also given an integer sequence $1 \leq a_1 \leq a_2 \leq \ldots \leq a_n$. Your goal is to fire the gun exactly a_n times, so that for all $1 \leq i \leq n$, the i-th robot gets hit by the ray gun exactly a_i times.

How many ways are there to achieve the goal? Two ways are different if you fire the gun a different number of times at time t, for some $t > 0$. We can show that the answer is finite, but it may be large, so output the remainder modulo 998 244 353.

Input

The first line contains n ($1 \leq n \leq 100$).

The second line contains n integers $a_1, a_2, \ldots a_n$ ($1 \leq a_1 \leq a_2 \leq \ldots \leq a_n \leq 100$).

Output

Print the answer, modulo 998 244 353.

Examples

standard input	standard output
1 1	1
5 1 1 1 2 2	2
5 100 100 100 100 100	1
2 2 5	84
6 1 2 5 6 8 8	22020096
4 2 3 20 25	846561846

Note

In the first test, the only solution is to fire the gun once at time $t = 1$.

In the second test, we again have to fire the gun once at $t = 1$. But we have 2 options for the second time we fire the gun. We can either fire a second time at $t = 6$ or $t = 7$. Note that we cannot fire between $2 \leq t \leq 5$, otherwise robot 3 would get hit a second time. And we cannot fire after $t \leq 8$, otherwise robot 4 would only get hit once.

In the third test, the only option is to fire the gun 100 times at $t = 1$.

Problem E. Even Even Odd Odd

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

Lunchbox has two arrays $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$, each of length n. It is guaranteed that $a_i \leq b_i$ for each $1 \leq i \leq n$.

He is allowed to perform either of the following two operations in any order, as many times as he wants (possibly zero):

- If there are an even number of even elements in a, add a positive even number to any element in a.
- If there are an odd number of odd elements in a, add a positive odd number to any element in a.

Is it possible to turn a into b?

Input

Input consists of multiple tests. The first line contains t, the number of tests ($1 \leq t \leq 10^5$).

The first line of each test contains n, the length of the arrays ($1 \leq n \leq 10^5$).

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq 10^9$).

The third line contains n integers $b_1, b_2, \ldots, b_n$ ($1 \leq b_i \leq 10^9$).

It is guaranteed that the sum of n across all tests does not exceed 10^5.

Output

For each test, output `YES` or `NO`.

Example

standard input	standard output
7 3 1 3 5 1 3 6 3 1 3 5 1 3 7 2 1 1 1 1 1 2 1000000000 5 1 2 3 4 5 6 7 8 9 10 5 1 1 1 1 1 3 3 3 3 4 2 1 2 1 4	YES YES YES NO NO YES NO

Note

In the first test, since there are initially 3 odd elements in a, we are allowed to add 1 (an odd integer) to a_3. After this operation, $a = [1, 3, 6] = b$.

In the second test, since there initially 0 even elements in a, we are allowed to add 2 (an even integer) to a_3. After this operation, $a = [1, 3, 7] = b$.

In the third test, it may be possible that $a = b$ without doing any operations.

In the fourth test, there are initially 0 odd elements and 1 even element, so we cannot perform any operations.

Problem F. Fill the Gym with Argon

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	1024 megabytes

Lacking the funds to fill the contest floor with argon, BAPC is looking for ways to raise money!

There are n fundraising activities that BAPC can hold, conveniently numbered 1 through n. Holding the i-th activity earns BAPC a profit of a_i dollars (this number can be positive, zero, or even negative). Furthermore, each activity $2 \leq i \leq n$ has a prerequisite r_i — meaning you cannot hold activity i without also holding activity r_i.

Of course, BAPC wants to maximize their profits, so they will choose a subset of activities that earns the maximum possible profit. If there are multiple such subsets, they choose a subset with the maximum number of activities (gotta look busy!).

So, the subset of activities that BAPC holds can be described by two integers: the profit made, and the number of activities held. We'll call these the *optimal profit*, and the *optimal number of activities* respectively.

But now the fundraising team wants to spice things up a bit by changing some a_i values. You are given q queries you need to answer, of two types:

- Type 1: given an integer v_i and value x_i, increase the profit of activity v_i by x_i — in other words, do $a_{v_i} := a_{v_i} + x_i$. **It is guaranteed that x_i is positive.**
- Type 2: given an integer v_i, find the smallest positive integer x_i such that increasing a_{v_i} by x_i will change either the optimal profit made, or the optimal number of activities held.

Type 1 updates are persistent, meaning that changes made to a_i remain in effect for future queries. Type 2 updates do **NOT** change the values of a_i.

Input

Input consists of multiple tests. The first line contains t, the number of tests ($1 \leq t \leq 1.5 \cdot 10^5$).

The first line of each test contains n — the number of activities ($2 \leq n \leq 3 \cdot 10^5$).

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($-10^9 \leq a_i \leq 10^9$).

The third line contains $n-1$ integers $r_2, r_3, \ldots, r_n$, the prerequisites ($1 \leq r_i \leq i-1$).

The fourth line contains q ($1 \leq q \leq 3 \cdot 10^5$).

The next q lines describe the queries. Each line starts with either a 1 or a 2.

- If the line starts with a 1, it is followed by two integers v_i and x_i ($1 \leq v_i \leq n$, $1 \leq x_i \leq 10^9$).
- If the line starts with a 2, it is followed by one integer v_i ($1 \leq v_i \leq n$).

It is guaranteed that the sum of n across all tests, and the sum of q across all tests both do not exceed $3 \cdot 10^5$.

Output

For each test, output the answers to each type 2 query.

Example

standard input	standard output
3 4 -2 0 0 0 1 1 2 3 2 3 1 1 10 2 1 4 100 -600 200 300 1 2 2 4 2 2 2 3 1 2 100 2 4 6 52 -42 6 5 -2 -93 1 2 3 2 1 8 1 3 76 1 5 35 2 5 2 6 1 6 94 1 1 20 1 2 25 2 4	2 1 100 100 1 1 93 1

Note

In the first test, the optimal strategy is initially not to hold any events at all. All events depend (directly or indirectly) on event 1, which loses 2 dollars. But no event makes any money to make up for it.

However, in the first query, if we were to change the profit of a_3 from 0 to 2, the optimal strategy would be to hold all events. The profit earned would still be 0, but we would now be holding 4 events, more than the 0 events we were holding before. Note that in a type 2 query, the change is purely hypothetical, and the values of a are not actually changed.

In the second query, we change a_1 from -2 into 8. The optimal strategy is now to host all events, earning an optimal profit of 8 dollars.

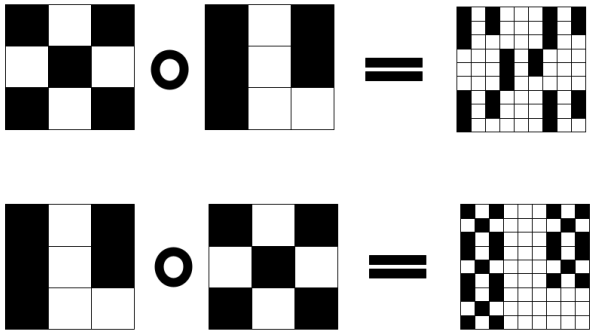
In the third query, we just need to change a_1 from 8 to 9 if we want to change the optimal profit. This is a difference of 1, so we output 1.

Problem G. Grids of Grids

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	1024 megabytes

Little Bobby Tables loves playing with tables. More specifically, he has a collection of m-by-m tables, where each cell is either empty or colored in.

For two tables T_1 and T_2, Bobby defines the *composition* of T_1 and T_2 (denoted $T_1 \circ T_2$) to be the grid obtained by replacing each colored-in cell in T_1 with a copy of T_2, and each blank cell with a m-by-m blank grid.



Examples of grid composition.

Two tables T_1 and T_2 are called *commutative* if $T_1 \circ T_2 = T_2 \circ T_1$. For example, the two tables in the picture above are not commutative, but two empty tables would be commutative.

You are given Bobby's collection of n tables $T_1, T_2, \ldots, T_n$. Count the number of pairs $1 \le i < j \le n$ where T_i and T_j are commutative.

In the examples below, tables are separated by an extra newline. These extra newlines are not present in the actual test data. They are only in the samples for ease of reading.

Input

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

The descriptions of Bobby's n tables follow. Each table is described by m lines of m characters each. Each character is either `.` (denoting an empty cell), or `X` (denoting a colored-in cell).

In the examples below, tables are separated by an extra newline. These extra newlines are not present in the actual test data. They are only in the samples for ease of reading.

Output

Print the number of commutative pairs.

Examples

standard input	standard output
3 3 X.X .X. X.X X.X X.X X.. X.X .X. X.X	1
4 5 XXXX. X...X XXXX. X...X XXXX. XXXXX X...X XXXXX X...X X...X XXXXX X...X XXXXX X.... X.... XXXXX X.... X.... X.... XXXXX	0
4 1 . X X .	6

Note

The first test corresponds to the pictures in the statement. Tables T_1 and T_2 are not commutative as shown above, and neither are T_2 and T_3. But T_1 and T_3 are equal, so $T_1 \circ T_3 = T_3 \circ T_1$. In total we have one commutative pair.

Problem H. Heaps of Queries

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

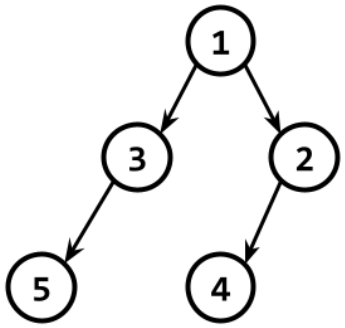
The *skew heap* data structure can be represented as a root node storing an integer value, and pointers to its left and right children. Each child is either another node, or an empty heap. We work with a simplified version, as follows.

If we want to insert an integer v into a heap H:

- If H is empty, then set H to a new node storing the value v, and no left or right children.
- Otherwise, swap the left and right children of the root. Then recursively insert v into the left child of the root.

Farmer John loves inserting elements into skew heaps. Every afternoon, he starts with an empty skew heap H. For all $i = 1, 2, \ldots, n$ in sequence, he then inserts i into H.

For example, here is a picture of the skew heap resulting from the insertions $1, 2, 3, 4, 5$:



Bessie, after watching this process, is curious about what the resulting heap will look like. She has q queries. In each one, she gives you an integer $1 \le x \le n$, and a string s consisting of characters `L`, `R`, and `U`, and needs you to answer the following question.

- Suppose you start at the node labeled x, and for each $i = 1, 2, \ldots, |s|$ in sequence, go to the left child of the current node if $s_i =$ `L`, go to the right child if $s_i =$ `R`, and go to the parent if $s_i =$ `U`. Tell Bessie the value stored in the node you end up at, or report that you go out of bounds of the heap during this process.

Input

The first line contains q, the number of Bessie's queries ($1 \le q \le 10^3$).

The only line of each query contains n, x, and s ($1 \le x \le n \le 10^9$, $1 \le |s| \le 100$).

It is guaranteed that each s consists of characters `L`, `R`, and `U`.

Output

For each query, output `-1` if you reach an empty heap, or the value stored in the ending node otherwise.

Example

standard input	standard output
14	3
5 1 L	2
5 1 R	5
5 1 LL	4
5 1 RL	-1
5 1 LLL	-1
5 1 U	1
5 4 UU	-1
5 2 RLLLRLRRLLLLLLLRRRR	-1
5 1 UL	-1
5 5 LU	-1
1 1 L	-1
1 1 U	-1
1 1 R	1605
696969 69 LRUURLLLRU	

Note

The first nine queries all have $n = 5$. Refer to the picture above.

We can see that taking paths `L`, `R`, `LL`, and `RL` from the root lead us to nodes 3, 2, 5, and 4 respectively. Taking the paths `RR` or `U`, though, will lead us outside of the heap, so we output `-1`.

Note that taking the path `UU` from node 4 will lead us $4 \to 2 \to 1$, so we output `1`.

In the ninth query, we go outside the heap after the first `U`, so we output `-1`.

Problem I. In the News

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	1024 megabytes

After all other jobs were lost to AI, Ezra decided to become a history teacher. To prepare his class for their future careers (also as history teachers), Ezra wants them to keep up with current events.

There will be n classes in the semester, and m news events. The i-th news event will take place between classes l_i and r_i inclusive. For each class, a subset of the students has already been chosen to present on news events. During that class, each of the chosen students has to present a news event.

Let's take a look at how students pick events to present. A student is characterized by their *enthusiasm* e_i. Just before a class when this student is supposed to present, suppose there are x news events that are currently ongoing and that have not been presented in a previous class. Then, this student will choose $\min(x, e_i)$ of these events, and be prepared to present any one of their chosen events.

During the i-th class, k_i students, with enthusiasm levels $e_1, e_2, \ldots, e_{k_i}$, will be presenting. Ezra can choose the order in which these students present. Each chosen student will choose and present one of the events they prepared **that has not been presented by another student**. But if a student's prepared topics have all already been presented by others, that student will become embarrassed and drop out of the class. Ezra doesn't want this to happen.

Now, Ezra turns to you for help. Is it possible for him to pick the order in which students present in each class, to guarantee that no student becomes embarrassed, no matter how they choose events?

In the examples below, tests are separated by an extra newline. These extra newlines are not present in the actual test data. They are only in the samples for ease of reading.

Input

Input consists of multiple tests. The first line contains t, the number of tests ($1 \le t \le 10^4$).

The first line of each test contains n and m, the number of classes and the number of events ($1 \le n \le m \le 2 \cdot 10^5$).

The next n lines each start with an integer k_i, representing the number of students presenting on day i ($1 \le k_i \le m$).

This is followed by k_i integers $e_1, e_2, \ldots, e_{k_i}$ on the same line — the enthusiasm levels of the students presenting that day ($1 \le e_i \le m$).

The next m lines contain two integers l_i and r_i each, the predicted start and end dates of the events ($1 \le l_i \le r_i \le n$).

It is guaranteed that the sum of k_i in each test does not exceed m. It is also guaranteed that the sum of m across all tests does not exceed $2 \cdot 10^5$.

In the examples below, tests are separated by an extra newline. These extra newlines are not present in the actual test data. They are only in the samples for ease of reading.

Output

For each test, output `YES` or `NO`.

Example

standard input	standard output
5 3 4 1 3 2 1 1 1 2 1 1 2 2 3 3 2 3 3 4 1 3 2 2 1 1 2 1 1 2 2 3 3 2 3 1 1 1 1 1 1 2 2 1 1 1 1 1 2 1 1 2 2 1 1 1 1 1 2 1 2	NO YES YES NO YES

Note

In the first test, no matter how Ezra orders the students during class 2, he cannot guarantee that no student will get embarrassed. For example, it is possible that both students presenting during class 2 pick event 4. Since they each only have one topic prepared, the second one to present will get embarrassed. In the second test, this is fixed — Ezra has the student with $e_i = 1$ present first. Then, no matter what topic the first student presents, the student with $e_i = 2$ always has a different topic ready to present.

In the third test, the only student just presents the only topic on the only day.

In the fourth test, it is possible that the student presenting in class 2 doesn't have any topic they can prepare! For example, suppose that the student presenting on day 1 chooses to present topic 1. Then, the student presenting on day 2 cannot present topic 1 since it was not presented before, and they cannot present topic 2 because that event ended after class 1. Since they have nothing to present, they become embarrassed.

In the fifth test, note that there may be events that start and end on the same days.

Problem J. Joystick Jumping

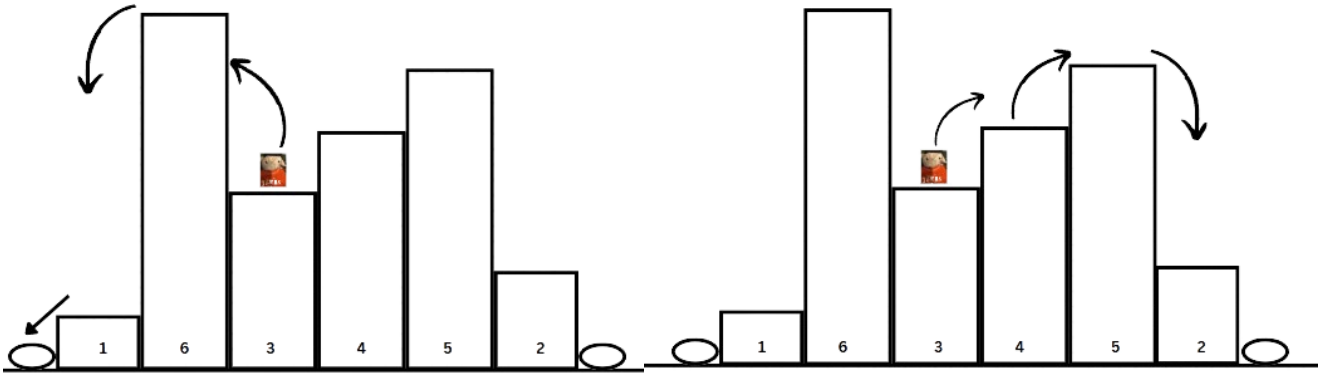
Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

Westin the sheep is playing a parkour game.

In the game, there are n buildings in a line, conveniently numbered 1 through n. The i-th building has height h_i. Westin spawns at building s, and his goal is to visit every building at least once.

Westin has an *energy level*, which is initially a non-negative integer m. In one move, he can jump from a building to an adjacent building. Jumping to a taller adjacent building costs 1 unit of energy, so you cannot jump to a taller adjacent building if your energy level is 0. Note that jumping to an adjacent building that is the same height or shorter height does not cost any energy, and is always allowed.

Finally, there are two teleporters in the game — one to the left of building 1, and one to the right of building n. Both of these teleporters are on the ground, so you can jump onto them for no energy cost, as long as you are on either building 1 or building n. Jumping on either teleporter takes you to the spawnpoint s, and restores your energy level to the original value, m.



A illustration of the optimal strategy in the last sample test. We first jump all the way to the left teleporter, then all the way to building n.

Now you know all the rules of the game. There's just one thing you don't know — where will Westin spawn? Solve the following problem for each $1 \le s \le n$:

- Suppose Westin spawns at building s. What's the minimum amount of energy he must start with if he wants to visit every building at least once?

Input

Input consists of multiple tests. The first line contains t, the number of tests ($1 \le t \le 10^4$).

The first line of each test contains n ($1 \le n \le 10^5$).

The second line contains n integers $h_1, h_2, \ldots, h_n$, the heights of the buildings ($1 \le h_i \le 10^9$).

It is guaranteed the sum of n across all tests does not exceed 10^5.

Output

In each test, output n integers. The i-th integer is the minimum energy you need to start with to be able to visit every building at least once, assuming you spawn at building i.

Example

standard input	standard output
5 5 1 1 1 1 1 4 1 2 3 4 5 1000000000 2 1 2 3 1 999 6 1 6 3 4 5 2	0 0 0 0 0 3 2 1 0 2 2 2 2 2 0 3 2 2 1 1 2

Note

In the first test, all buildings are equal height, so you can jump from any building to any adjacent building with 0 cost.

In the second test, the optimal strategy from building 3 is to use 1 energy to jump to building 4, then jump to building 3, then jump to building 2, then jump to building 1. In total we use 1 energy. Note, in particular, we don't need to end up at the same building as we start at.

Problem K. Killer Cows

Input file:	`standard input`
Output file:	`standard output`
Time limit:	10 seconds
Memory limit:	1024 megabytes

Farmer John has n cows that he needs to ferry across a river. He has a boat which can take at most k cows across at a time. Farmer John will repeatedly ride the boat across the river with some (potentially empty) set of cows until all cows are on the other side of the river.

There are m sets of cows that cannot be left alone, otherwise they will plan the violent seizure of the farm from their human master. Specifically, for each of these subsets of cows, whenever FJ is riding the boat, they cannot all be on the same side of the river (even if they are with other cows).

Find the minimum number of times FJ needs to ride the boat to ferry all cows across the river, or report that it is impossible.

Input

The first line contains n, m, and k ($1 \le k \le n \le 20$, $0 \le m \le 10^5$).

The next m lines each contain a string of length n, consisting of `0`s and `1`s, representing a subset of cows. The i-th cow is in a subset if and only if the i-th character of that string is a `1`.

It is guaranteed that all m subsets are distinct.

It is also guaranteed that all subsets are nonempty, and no subset is the entire set of cows.

Output

If it is impossible to ferry all the cows to the other side of the river, output `-1`.

Otherwise print a single integer, the minimum number of trips needed.

Examples

standard input	standard output
7 0 2	7
7 7 2 1000000 0100000 0010000 0001000 0000100 0000010 0000001	-1
7 7 7 1000000 0100000 0010000 0001000 0000100 0000010 0000001	1
3 2 1 110 011	7

Note

In the first test, FJ can:

- Take cows 1 and 2 to the other side.
- Return to the starting side.
- Take cows 3 and 4 to the other side.
- Return to the starting side.
- Take cows 5 and 6 to the other side.
- Return to the starting side.
- Take cow 7 the other side.

In total, it takes 7 trips. We can show this is optimal.

In the second test, the goal is impossible, since no matter what, FJ must leave at least 5 cows on the starting side on his first trip. But no cow can be left on a shore.

In the third test, FJ can:

- Take cow 2 to the other side.
- Return to the starting side.
- Take cow 1 to the other side.
- Take cow 2 back to the starting side.
- Take cow 3 to the other side.
- Return to the starting side.
- Take cow 2 to the other side for a final time.

In total, it takes 7 trips. We can show this is optimal.