Codeforces Round #767 (Div. 2)

A. Download More RAM

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

Did you know you can download more RAM? There is a shop with n different pieces of software that increase your RAM. The i-th RAM increasing software takes a_i GB of memory to run (**temporarily, once the program is done running, you get the RAM back**), and gives you an additional b_i GB of RAM (permanently). **Each software can only be used once.** Your PC currently has k GB of RAM.

Note that you can't use a RAM-increasing software if it takes more GB of RAM to use than what you currently have.

Since RAM is the most important thing in the world, you wonder, what is the maximum possible amount of RAM achievable?

Input
The first line of the input contains a single integer t ($1 \leq t \leq 100$) — the number of test cases. The description of test cases follows.

The first line of each test case contains the integers n and k ($1 \leq n \leq 100$, $1 \leq k \leq 1000$). Then two lines follow, each containing n integers describing the arrays a and b ($1 \leq a_i, b_i \leq 1000$).

Output
For each test case, output a single line containing the largest amount of RAM you can achieve.

Example

input
4 3 10 20 30 10 9 100 10 5 1 1 1 5 1 1 1 1 1 1 1 5 1 2 2 2 2 2 100 100 100 100 100 5 8 128 64 32 16 8 128 64 32 16 8
output
29 6 1 256

Note
In the first test case, you only have enough RAM to run the third software initially, but that increases your RAM to 20 GB, which allows you to use the first software, increasing your RAM to 29 GB. The only software left needs 30 GB of RAM, so you have to stop here.

In the second test case, you can use the first, second, fourth and fifth software that need only 1 GB of RAM per software to run to increase your RAM to 5 GB, and then use the last remaining one to increase your RAM to 6 GB.

In the third test case, all the software need more than 1 GB of RAM to run, so the amount of RAM you have stays at 1 GB.

B. GCD Arrays

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

Consider the array a composed of all the integers in the range $[l, r]$. For example, if $l = 3$ and $r = 7$, then $a = [3, 4, 5, 6, 7]$.

Given l, r, and k, is it possible for $\gcd(a)$ to be greater than 1 after doing the following operation at most k times?

- Choose 2 numbers from a.
- Permanently remove one occurrence of each of them from the array.
- Insert their product back into a.

$\gcd(b)$ denotes the greatest common divisor (GCD) of the integers in b.

Input
The first line of the input contains a single integer t ($1 \le t \le 10^5$) — the number of test cases. The description of test cases follows.

The input for each test case consists of a single line containing 3 non-negative integers l, r, and k ($1 \le l \le r \le 10^9$, $0 \le k \le r - l$).

Output
For each test case, print "YES" if it is possible to have the GCD of the corresponding array greater than 1 by performing at most k operations, and "NO" otherwise (case insensitive).

Example

input
9 1 1 0 3 5 1 13 13 0 4 4 0 3 7 4 4 10 3 2 4 0 1 7 3 1 5 3
output
NO NO YES YES YES YES NO NO YES

Note
For the first test case, $a = [1]$, so the answer is "NO", since the only element in the array is 1.

For the second test case the array is $a = [3, 4, 5]$ and we have 1 operation. After the first operation the array can change to: $[3, 20]$, $[4, 15]$ or $[5, 12]$ all of which having their greatest common divisor equal to 1 so the answer is "NO".

For the third test case, $a = [13]$, so the answer is "YES", since the only element in the array is 13.

For the fourth test case, $a = [4]$, so the answer is "YES", since the only element in the array is 4.

C. Meximum Array

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

Mihai has just learned about the MEX concept and since he liked it so much, he decided to use it right away.

Given an array a of n non-negative integers, Mihai wants to create **a new array** b that is formed in the following way:

While a is not empty:

- Choose an integer k ($1 \le k \le |a|$).
- Append the MEX of the first k numbers of the array a to the end of array b and erase them from the array a, shifting the positions of the remaining numbers in a.

But, since Mihai loves big arrays as much as the MEX concept, he wants the new array b to be the **lexicographically maximum**. So, Mihai asks you to tell him what the maximum array b that can be created by constructing the array optimally is.

An array x is lexicographically greater than an array y if in the first position where x and y differ $x_i > y_i$ or if $|x| > |y|$ and y is a prefix of x (where $|x|$ denotes the size of the array x).

The **MEX** of a set of non-negative integers is the minimal non-negative integer such that it is not in the set. For example, **MEX**($\{1, 2, 3\}$) $= 0$ and **MEX**($\{0, 1, 2, 4, 5\}$) $= 3$.

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

The first line of each test case contains a single integer n ($1 \le n \le 2 \cdot 10^5$) — the number of elements in the array a.

The second line of each test case contains n **non-negative** integers $a_1, \ldots, a_n$ ($0 \le a_i \le n$), where a_i is the i-th integer from the array a.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output
For each test case print m — the length of the maximum array b Mihai can create, followed by m integers denoting the elements of the array b.

Example

input
6 5 1 0 2 0 3 8 2 2 3 4 0 1 2 0 1 1 5 0 1 2 3 4 4 0 1 1 0 10 0 0 2 1 1 1 0 0 1 1
output
1 4 2 5 1 1 0 1 5 2 2 2 4 3 2 2 0

Note
In the first test case, the lexicographically maximum array b is obtained by selecting $k = 5$, resulting in the MEX of the whole array a. It is lexicographically maximum because an array starting with a smaller number than 4 is lexicographically smaller, and choosing a $k < 5$ would result in an array starting with a number smaller than 4.

In the second test case, there are two ways to obtain the maximum array: first selecting $k = 6$, then $k = 2$, or first selecting $k = 7$ and then $k = 1$.

D. Peculiar Movie Preferences

time limit per test: 2 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

Mihai plans to watch a movie. He only likes palindromic movies, so he wants to skip some (possibly zero) scenes to make the remaining parts of the movie palindromic.

You are given a list s of n non-empty strings of length **at most** 3, representing the scenes of Mihai's movie.

A subsequence of s is called *awesome* if it is non-empty and the concatenation of the strings in the subsequence, in order, is a palindrome.

Can you help Mihai check if there is at least one awesome subsequence of s?

A palindrome is a string that reads the same backward as forward, for example strings "z", "aaa", "aba", "abccba" are palindromes, but strings "codeforces", "reality", "ab" are not.

A sequence a is a non-empty subsequence of a non-empty sequence b if a can be obtained from b by deletion of several (possibly zero, but not all) elements.

Input
The first line of the input contains a single integer t ($1 \leq t \leq 100$) — the number of test cases. The description of test cases follows.

The first line of each test case contains a single integer n ($1 \leq n \leq 10^5$) — the number of scenes in the movie.

Then follows n lines, the i-th of which containing a single non-empty string s_i of length at most 3, consisting of lowercase Latin letters.

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

Output
For each test case, print "YES" if there is an awesome subsequence of s, or "NO" otherwise (case insensitive).

Example

input
6 5 zx ab cc zx ba 2 ab bad 4 co def orc es 3 a b c 3 ab cd cba 2 ab ab
output
YES NO NO YES YES NO

Note
In the first test case, an awesome subsequence of s is $[ab, cc, ba]$

E. Grid Xor

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

Note: The XOR-sum of set $\{s_1, s_2, \ldots, s_m\}$ is defined as $s_1 \oplus s_2 \oplus \ldots \oplus s_m$, where $\oplus$ denotes the bitwise XOR operation.

After almost winning IOI, Victor bought himself an $n \times n$ grid containing integers in each cell. n **is an even integer.** The integer in the cell in the i-th row and j-th column is $a_{i,j}$.

Sadly, Mihai stole the grid from Victor and told him he would return it with only one condition: Victor has to tell Mihai the XOR-sum of **all** the integers in the whole grid.

Victor doesn't remember all the elements of the grid, but he remembers some information about it: For each cell, Victor remembers the XOR-sum of all its neighboring cells.

Two cells are considered neighbors if they share an edge — in other words, for some integers $1 \leq i, j, k, l \leq n$, the cell in the i-th row and j-th column is a neighbor of the cell in the k-th row and l-th column if $|i - k| = 1$ and $j = l$, or if $i = k$ and $|j - l| = 1$.

To get his grid back, Victor is asking you for your help. Can you use the information Victor remembers to find the XOR-sum of the whole grid?

It can be proven that the answer is unique.

Input
The first line of the input contains a single integer t ($1 \leq t \leq 100$) — the number of test cases. The description of test cases follows.

The first line of each test case contains a single **even** integer n ($2 \leq n \leq 1000$) — the size of the grid.

Then follows n lines, each containing n integers. The j-th integer in the i-th of these lines represents the XOR-sum of the integers in all the neighbors of the cell in the i-th row and j-th column.

It is guaranteed that the sum of n over all test cases doesn't exceed 1000 and in the original grid $0 \leq a_{i,j} \leq 2^{30} - 1$.

Hack Format

To hack a solution, use the following format:

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

The first line of each test case should contain a single **even** integer n ($2 \leq n \leq 1000$) — the size of the grid.

Then n lines should follow, each containing n integers. The j-th integer in the i-th of these lines is $a_{i,j}$ in Victor's **original** grid. The values in the grid should be integers in the range $[0, 2^{30} - 1]$

The sum of n over all test cases must not exceed 1000.

Output
For each test case, output a single integer — the XOR-sum of the whole grid.

Example

input
3 2 1 5 5 1 4 1 14 8 9 3 1 5 9 4 13 11 1 1 15 4 11 4 2 4 1 6 3 7 3 10 15 9 4 2 12 7 15 1
output
4 9 5

Note
For the first test case, one possibility for Victor's original grid is:

1	3
2	4

For the second test case, one possibility for Victor's original grid is:

3	8	8	5
9	5	5	1
5	5	9	9
8	4	2	9

For the third test case, one possibility for Victor's original grid is:

4	3	2	1
1	2	3	4
5	6	7	8
8	9	9	1

F1. Game on Sum (Easy Version)

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

This is the easy version of the problem. The difference is the constraints on n, m and t. You can make hacks only if all versions of the problem are solved.

Alice and Bob are given the numbers n, m and k, and play a game as follows:

The game has a score that Alice tries to maximize, and Bob tries to minimize. The score is initially 0. The game consists of n turns. Each turn, Alice picks a **real** number from 0 to k (inclusive) which Bob either adds to or subtracts from the score of the game. But throughout the game, Bob has to choose to add at least m out of the n turns.

Bob gets to know which number Alice picked before deciding whether to add or subtract the number from the score, and Alice gets to know whether Bob added or subtracted the number for the previous turn before picking the number for the current turn (except on the first turn since there was no previous turn).

If Alice and Bob play optimally, what will the final score of the game be?

Input
The first line of the input contains a single integer t ($1 \le t \le 1000$) — the number of test cases. The description of test cases follows.

Each test case consists of a single line containing the three integers, n, m, and k ($1 \le m \le n \le 2000, 0 \le k < 10^9 + 7$) — the number of turns, how many of those turns Bob *has to* add, and the biggest number Alice can choose, respectively.

It is guaranteed that the sum of n over all test cases does not exceed 2000.

Output
For each test case output a single **integer** number — the score of the optimal game modulo $10^9 + 7$.

Formally, let $M = 10^9 + 7$. It can be shown that the answer can be expressed as an irreducible fraction $\frac{p}{q}$, where p and q are integers and $q \not\equiv 0 \pmod{M}$. Output the integer equal to $p \cdot q^{-1} \bmod M$. In other words, output such an integer x that $0 \le x < M$ and $x \cdot q \equiv p \pmod{M}$.

Example

input
7 3 3 2 2 1 10 6 3 10 6 4 10 100 1 1 4 4 0 69 4 20
output
6 5 375000012 500000026 958557139 0 49735962

Note
In the first test case, the entire game has 3 turns, and since $m = 3$, Bob has to add in each of them. Therefore Alice should pick the biggest number she can, which is $k = 2$, every turn.

In the third test case, Alice has a strategy to guarantee a score of $\frac{75}{8} \equiv 375000012 \pmod{10^9 + 7}$.

In the fourth test case, Alice has a strategy to guarantee a score of $\frac{45}{2} \equiv 500000026 \pmod{10^9 + 7}$.

F2. Game on Sum (Hard Version)

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

This is the hard version of the problem. The difference is the constraints on n, m and t. You can make hacks only if all versions of the problem are solved.

Alice and Bob are given the numbers n, m and k, and play a game as follows:

The game has a score that Alice tries to maximize, and Bob tries to minimize. The score is initially 0. The game consists of n turns. Each turn, Alice picks a **real** number from 0 to k (inclusive) which Bob either adds to or subtracts from the score of the game. But throughout the game, Bob has to choose to add at least m out of the n turns.

Bob gets to know which number Alice picked before deciding whether to add or subtract the number from the score, and Alice gets to know whether Bob added or subtracted the number for the previous turn before picking the number for the current turn (except on the first turn since there was no previous turn).

If Alice and Bob play optimally, what will the final score of the game be?

Input
The first line of the input contains a single integer t ($1 \le t \le 10^5$) — the number of test cases. The description of test cases follows.

Each test case consists of a single line containing the three integers, n, m, and k ($1 \le m \le n \le 10^6, 0 \le k < 10^9 + 7$) — the number of turns, how many of those turns Bob *has to* add, and the biggest number Alice can choose, respectively.

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

Output

For each test case output a single **integer** number — the score of the optimal game modulo $10^9 + 7$.

Formally, let $M = 10^9 + 7$. It can be shown that the answer can be expressed as an irreducible fraction $\frac{p}{q}$, where p and q are integers and $q \not\equiv 0 \pmod{M}$. Output the integer equal to $p \cdot q^{-1} \bmod M$. In other words, output such an integer x that $0 \le x < M$ and $x \cdot q \equiv p \pmod{M}$.

Example

input
7 3 3 2 2 1 10 6 3 10 6 4 10 100 1 1 4 4 0 69 4 20
output
6 5 375000012 500000026 958557139 0 49735962

Note

In the first test case, the entire game has 3 turns, and since $m = 3$, Bob has to add in each of them. Therefore Alice should pick the biggest number she can, which is $k = 2$, every turn.

In the third test case, Alice has a strategy to guarantee a score of $\frac{75}{8} \equiv 375000012 \pmod{10^9 + 7}$.

In the fourth test case, Alice has a strategy to guarantee a score of $\frac{45}{2} \equiv 500000026 \pmod{10^9 + 7}$.