A. 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 $(1 \leq k \leq |a|)$.
- 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 $(1 \leq t \leq 100)$ — the number of test cases. The description of test cases follows.

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

The second line of each test case contains n **non-negative** integers $a_1, \ldots, a_n$ $(0 \leq a_i \leq n)$, 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 $k = 1$.

B. 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 $(1 \le t \le 100)$ — the number of test cases. The description of test cases follows.

The first line of each test case contains a single integer n $(1 \le n \le 10^5)$ — 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 $[ab, cc, ba]$

C. 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 $1 \le i, j, k, l \le n$, 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 ($1 \le t \le 100$) — the number of test cases. The description of test cases follows.

The first line of each test case contains a single **even** integer n ($2 \le n \le 1000$) — 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 $0 \le a_{i,j} \le 2^{30} - 1$.

Hack Format

To hack a solution, use the following format:

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

The first line of each test case should contain a single **even** integer n ($2 \le n \le 1000$) — 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

D1. 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 $\frac{45}{2} \equiv 500000026 \pmod{10^9+7}$.

D2. 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 $\frac{45}{2} \equiv 500000026 \pmod{10^9+7}$.

E. Groceries in Meteor Town

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

Mihai lives in a town where meteor storms are a common problem. It's annoying, because Mihai has to buy groceries sometimes, and getting hit by meteors isn't fun. Therefore, we ask you to find the most dangerous way to buy groceries so that we can trick him to go there.

The town has n buildings numbered from 1 to n. Some buildings have roads between them, and there is exactly 1 simple path from any building to any other building. Each road has a certain meteor danger level. The buildings all have grocery stores, but Mihai only cares about the open ones, of course. Initially, all the grocery stores are closed.

You are given q queries of three types:

1. Given the integers l and r, the buildings numbered from l to r open their grocery stores (nothing happens to buildings in the range that already have an open grocery store).
2. Given the integers l and r, the buildings numbered from l to r close their grocery stores (nothing happens to buildings in the range that didn't have an open grocery store).
3. Given the integer x, find the maximum meteor danger level on the simple path from x to **any** open grocery store, or -1 if there is no edge on any simple path to an open store.

Input

The first line contains the two integers n and q ($2 \le n, q \le 3 \cdot 10^5$).

Then follows $n-1$ lines, the i-th of which containing the integers u_i, v_i, and w_i ($1 \le u_i, v_i \le n,\ \ 1 \le w_i \le 10^9$) meaning there is two way road between building u_i and v_i with meteor danger level w_i.

It is guaranteed that the given edges form a tree.

Then follows q lines, the j-th of which begin with the integer t_j ($1 \le t_j \le 3$), meaning the j-th query is of the t_j-th type.

If t_j is 1 or 2 the rest of the line contains the integers l_j and r_j ($1 \le l_j \le r_j \le n$).

If t_j is 3 the rest of the line contains the integer x_j ($1 \le x_j \le n$).

Output

For each query of the 3rd type ($t_j = 3$), output the maximum meteor danger level that is on some edge on the simple path from x_j to some open store, or -1 if there is no such edge.

Example

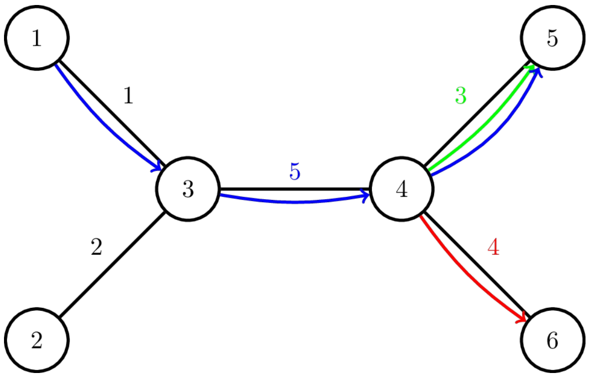
input

```
6 9
1 3 1
2 3 2
4 5 3
4 6 4
3 4 5
3 1
1 1 1
3 1
2 1 1
1 5 6
3 4
2 6 6
3 4
3 1
```

output

```
-1
-1
4
3
5
```

Note



This is an illustration of the town given in the sample input.

In the first query, there are no open stores, so obviously there are no edges on the simple path from 1 to any open store, so the answer is -1.

After the second and third queries, the set of open stores is $\{1\}$. The simple path from 1 to 1 has no edges, so the answer for the 3 rd query is -1.

After the fourth query, there are no open stores.

After the fifth and sixth queries, the set of open stores is $\{5, 6\}$. In the sixth query, there are two paths from $x_j = 4$ to some open grocery store: 4 to 5 and 4 to 6. The biggest meteor danger is found on the edge from 4 to 6, so the answer for the 6th query is 4. This path is marked with red in the illustration.

After the rest of the queries, the set of open stores is $\{5\}$. In the eighth query, the only path from $x_j = 4$ to an open store is from 4 to 5, and the maximum weight on that path is 3. This path is marked with green in the illustration.

In the ninth query, the only path from $x_j = 1$ to an open store is from 1 to 5, and the maximum weight on that path is 5. This path is marked with blue in the illustration.

F. Spaceship Crisis Management

time limit per test: 8 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

NASA (Norwegian Astronaut Stuff Association) is developing a new steering system for spaceships. But in its current state, it wouldn't be very safe if the spaceship would end up in a bunch of space junk. To make the steering system safe, they need to answer the following:

Given the target position $t = (0, 0)$, a set of n pieces of space junk l described by line segments $l_i = ((a_{ix}, a_{iy}), (b_{ix}, b_{iy}))$, and a starting position $s = (s_x, s_y)$, is there a direction such that floating in that direction from the starting position would lead to the target position?

When the spaceship hits a piece of space junk, what happens depends on the absolute difference in angle between the floating direction and the line segment, θ:

- If $\theta < 45^\circ$, the spaceship slides along the piece of space junk in the direction that minimizes the change in angle, and when the spaceship slides off the end of the space junk, it continues floating in the direction it came in (before hitting the space junk).
- If $\theta \geq 45^\circ$, the spaceship stops, because there is too much friction to slide along the space junk.

You are only given the set of pieces of space junk once, and the target position is always $(0, 0)$, but there are q queries, each with a starting position $s_j = (s_{jx}, s_{jy})$.

Answer the above question for each query.

Input

The first line contains the the integer n ($1 \leq n \leq 1500$).

Then follows n lines, the i-th of which containing the 4 integers a_{ix}, a_{iy}, b_{ix}, and b_{iy} ($|a_{ix}|, |a_{iy}|, |b_{ix}|, |b_{iy}| \leq 1000$).

Then follows a line containing the integer q ($1 \leq q \leq 1000$).

Then follows q lines, the j-th of which containing the 2 integers s_{jx} and s_{jy} ($|s_{jx}|, |s_{jy}| \leq 1000$).

It is guaranteed that none of the segments in l cross or touch, that t is not on any segment in l, that s_j is not on any segment in l, and that $s \neq t$.

Output

For each query s_j print an answer. If there exists a direction such that floating from s_j in that direction, possibly sliding along some pieces of space junk, leads to t, print "YES". Otherwise, print "NO" (case insensitive).

Example

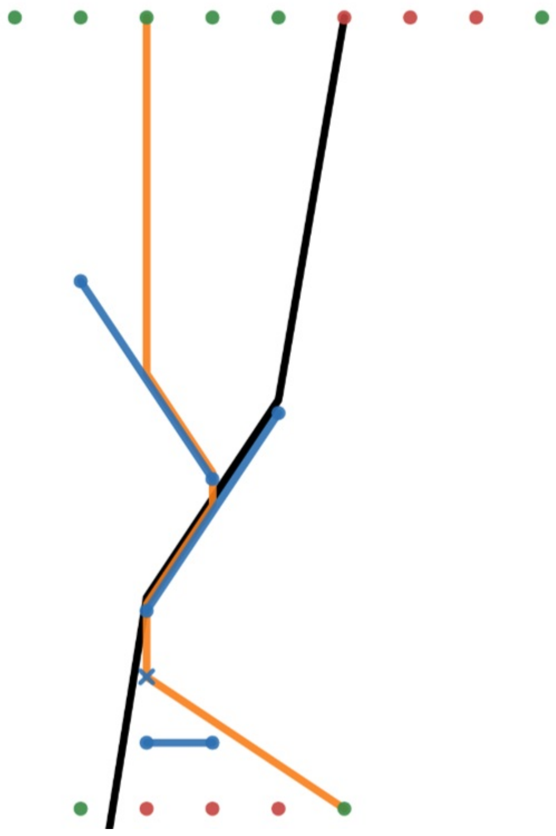
input

```
3
0 1 2 4
1 3 -1 6
0 -1 1 -1
14
-2 10
-1 10
0 10
1 10
2 10
3 10
4 10
5 10
6 10
-1 -2
0 -2
1 -2
2 -2
3 -2
```

output

```
YES
YES
YES
YES
YES
NO
NO
NO
YES
YES
NO
NO
NO
YES
```

Note



The blue cross represents the target location, and the other blue line segments represent the space junk.

Green dots represent starting locations where the answer is yes, and red dots represent starting locations where the answer is no.

The yellow lines are possible paths to the target location for the 3rd and 14th queries.

The black line is a possible path from the starting location in the 6th query, but it barely misses the target location.