Problem A: App Versioning

With the advancement of AI, writing code is now much faster than ever before. Many AI-pilled companies estimate that the number of lines of code they will produce in 2026 will be at least 20 times that of 2025. More code means more releases, and app versioning will become much more complicated. To prepare for this future, you are building a new versioning system.

A *version* is a sequence of one or more non-negative integers, called *components*, separated by dots. For example, `1.2.3`, `2.1` and `10` are versions. Each component must be written in decimal **without leading zeros**. In particular, `0` is a valid component, but `02` is not, so `1.02.3` is not a valid version.

Unlike the versions used by humans today, which usually have at most four components, versions in your system can have hundreds of thousands of components. This is obviously not an issue for a system whose target audience is AI agents.

Two versions are compared component by component, from left to right:

- At the first position where the components of the two versions are different, the version whose component is smaller **as an integer** is the smaller version. For example, `1.2.3` < `1.2.4`, `1.9` < `1.10` and `1.3` < `2.1`.
- If there is no such position, i.e., one version is a prefix of the other, the version with fewer components is the smaller version. For example, `1.3` < `1.3.1` and `1.3` < `1.3.0`.

Unfortunately, when testing your system, you found two versions where the dots have been lost. More precisely, you have two strings of digits A and B. You need to insert dots into A and into B to turn them back into versions V_A and V_B. A dot can be inserted between any two adjacent digits, and at most one dot can be inserted between each pair of adjacent digits. Inserting no dots at all is also allowed. The digits themselves cannot be changed, removed or reordered.

Count the number of ways to insert dots such that both V_A and V_B are valid versions and $V_A < V_B$. Two ways are different if the resulting pairs (V_A, V_B) are different. Since the answer can be very large, print it modulo $10^9 + 7$.

Input

Each test contains multiple test cases. The first line contains the number of test cases τ ($1 \le \tau \le 5 \times 10^5$). The description of the test cases follows.

Each test case consists of a single line containing two strings A and B ($1 \le |A|, |B| \le 5 \cdot 10^5$), separated by a space. Both strings consist only of digits `0`–`9`.

It is guaranteed that the sum of $|A| + |B|$ over all test cases does not exceed 10^6.

Output

For each test case, print the number of ways modulo $10^9 + 7$.

Sample Input 1	Sample Output 1
3 12 13 102 11 0 00	3 3 1

Sample Explanation

In the first test case, V_A can be `12` or `1.2`, and V_B can be `13` or `1.3`. There are 3 ways such that $V_A < V_B$: `12` $<$ `13`, `1.2` $<$ `13` and `1.2` $<$ `1.3`.

In the second test case, V_A can be `102`, `10.2` or `1.0.2` (but not `1.02`, which has a leading zero), and V_B can be `11` or `1.1`. There are 3 ways such that $V_A < V_B$: `10.2` $<$ `11`, `1.0.2` $<$ `11` and `1.0.2` $<$ `1.1`.

In the third test case, the only valid way is $V_A =$ `0` and $V_B =$ `0.0`, and indeed `0` $<$ `0.0`.

Problem B: Building Boundaries

Bash is developing an online city-building game. The map is the Cartesian plane with n towers placed on it. The i-th tower is located at the point (x_i, y_i).

During the game, the players build m walls one after another. The i-th wall is a straight segment connecting tower u_i and tower v_i. The walls are built so that:

- two walls never intersect, except possibly at a common endpoint;
- a wall does not pass through any tower other than its two endpoints;
- no tower lies **strictly inside** any polygon whose sides are walls.

A contiguous region on the plane is *covered* if every point inside the region also lies inside a polygon whose sides are the walls that have been built. A *maximal region* is a covered contiguous region that is also not strictly contained in any other covered contiguous region.

To show the players their progress, the game needs to display the total area of all maximal regions after each wall is built. Help Bash compute these values.

Input

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

- The first line of each test case contains two integers n and m ($2 \le n \le 2 \cdot 10^5$; $1 \le m \le 2 \cdot 10^5$) – the number of towers and the number of walls.
- The i-th of the next n lines contains two integers x_i and y_i ($0 \le x_i, y_i \le 10^9$) – the coordinates of the i-th tower. All towers are at distinct points.
- The i-th of the next m lines contains two integers u_i and v_i ($1 \le u_i, v_i \le n$; $u_i \ne v_i$) – the towers connected by the i-th wall, in the order the walls are built. No two walls connect the same pair of towers. The walls satisfy the conditions in the statement.

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

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

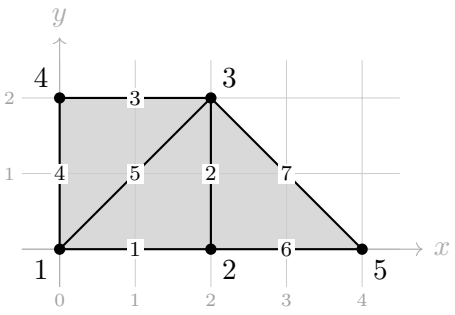
Output

For each test case, print a line with m integers. The i-th integer is **twice** the sum of areas of all maximal regions after the first i walls are built. It can be shown that this value is always an integer.

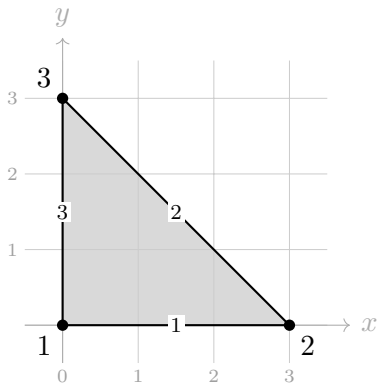
Sample Input 1	Sample Output 1
2 5 7 0 0 2 0 2 2 0 2 4 0 1 2 2 3 3 4 4 1 1 3 2 5 5 3 3 3 0 0 3 0 0 3 1 2 2 3 3 1	0 0 0 8 8 8 12 0 0 9

Sample Explanation

The figure below shows both test cases after all walls are built. All maximal covered regions are shaded, and each wall is labeled with the order in which it is built.



First test case



Second test case

In the first test case, after the first 4 walls, the towers 1, 2, 3 and 4 form a square of area 4, so the answer is 8. The 5-th wall is a diagonal of the square, which does not change the maximal covered region. The 6-th wall does not form any new polygon. The 7-th wall forms a new triangle with towers 2, 5 and 3, of area 2. The total maximal covered area is 6, so the answer is 12.

In the second test case, the maximal covered region after the last wall is a triangle of area 4.5, so the answer is 9.

Problem C: Colorful Candies

You have a huge jar of magic candies of n different colors. The colors are numbered from 1 to n, and initially the i-th color has c_i candies.

While the candies in the jar are not all of the same color, you perform the following operation:

- Choose two candies of **different** colors, uniformly at random among all such pairs of candies.
- Then, choose one of these two candies, uniformly at random, and change its color to the color of the other candy.

Find the expected number of operations performed until all the candies in the jar have the same color.

Input

Each test contains multiple test cases. The first line contains the number of test cases τ ($1 \le \tau \le 300$). The description of the test cases follows.

The first line of each test case contains an integer n ($1 \le n \le 300$) – the number of colors.

The second line contains n integers $c_1, c_2, \ldots, c_n$ ($1 \le c_i \le 10^9$) – the number of candies of each color.

It is guaranteed that the sum of n over all test cases does not exceed 300.

Output

For each test case, print the answer as follows:

- If the expected number of operations is infinite, print -1.
- Otherwise, let $M = 10^9 + 7$. It can be shown that in this case, the expected number of operations 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 $x \cdot q \equiv p \pmod{M}$.

Sample Input 1	Sample Output 1
3 2 1 1 2 2 1 1 5	1 2 0

Sample Explanation

In the first test case, there are two candies of different colors. After one operation, both candies have the same color, so the answer is 1.

In the second test case, there are two candies of color 1 and one candy of color 2. In the first operation, the chosen pair consists of the candy of color 2 and one candy of color 1:

- With probability $\frac{1}{2}$, the candy of color 2 changes to color 1, and all candies have the same color.
- With probability $\frac{1}{2}$, the candy of color 1 changes to color 2. Now there is one candy of color 1 and two candies of color 2, which is the same situation as the beginning with the colors swapped.

Hence, the number of operations follows a geometric distribution with success probability $\frac{1}{2}$. Consequently, the expected number of operations is 2.

In the third test case, all candies already have the same color, so no operation is performed.

Problem D: Delightful Ordering

The Kingdom of Exordia has n cities, labeled from 0 to $n-1$. The cities are connected by $n-1$ bidirectional roads such that it is possible to travel between any pair of cities using one or more of these roads.

Queen ERB, the ruler of the kingdom, goes on an incognito tour every year to learn about the everyday life of her citizens. She always travels according to the following rules.

Initially, Queen ERB is at the capital (city 0), which is considered visited. Then, she repeats the following steps:

- If there is at least one unvisited city directly connected to her current city, she travels to the one with the smallest label among them. That city becomes visited.
- Otherwise, if she is in the capital, the tour ends.
- Otherwise, she travels back to the city from which she arrived at her current city for the first time.

The *first-visit sequence* is the sequence of cities in the order they are visited for the first time. Note that it always starts with city 0 and contains every city exactly once.

To improve the traffic in the kingdom, Queen ERB has decided to build exactly one road between a pair of cities that is not directly connected yet. However, her royal guards have already planned their secret escort around her current route, so she wants the first-visit sequence to stay exactly the same after the new road is built.

You are entrusted by the Queen to count the number of ways to build the new road such that the first-visit sequence remains unchanged. Two ways are considered different if the new road connects a different pair of cities. Can you help her?

Input

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

- The first line of each test case contains an integer n ($3 \le n \le 2 \times 10^5$) – the number of cities.
- The i-th of the next $n-1$ lines contains two integers u_i and v_i ($0 \le u_i, v_i < n$, $u_i \ne v_i$) – the cities connected by the i-th road. It is guaranteed that the given roads connect all n cities.

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

Output

For each test case, print a single integer – the number of ways.

Sample Input 1	Sample Output 1
2 4 0 2 2 1 2 3 5 0 2 0 3 3 1 3 4	2 3

Sample Explanation

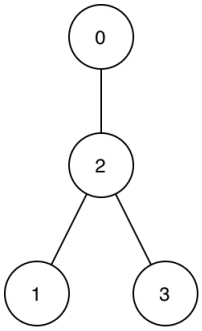


Figure D.1: The first test case

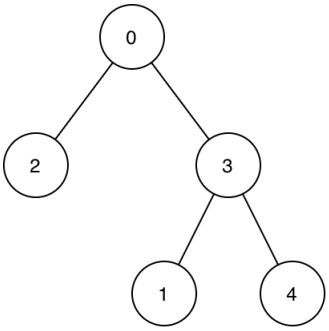


Figure D.2: The second test case

In the first test case, Queen ERB visits cities in the following order: $[0, 2, 1, 2, 3, 2, 0]$ so the first-visit sequence is $[0, 2, 1, 3]$. There are 3 pairs of cities that are not directly connected: $(0, 1)$, $(0, 3)$ and $(1, 3)$.

- If the road $(0, 1)$ is built, the new first-visit sequence is $[0, 1, 2, 3]$.
- If the road $(0, 3)$ is built, the new first-visit sequence is $[0, 2, 1, 3]$.
- If the road $(1, 3)$ is built, the new first-visit sequence is $[0, 2, 1, 3]$.

Therefore, there are 2 ways that keep the first-visit sequence unchanged: $(0, 3)$ and $(1, 3)$.

In the second test case, there are 3 ways to build a road without changing the first-visit sequence: $(0, 4)$, $(1, 4)$, $(2, 3)$.

Problem E: Easy Math

Given an integer k, consider two three-digit numbers X and Y such that:

- the six digits of X and Y are pairwise distinct;
- the hundreds digit of Y is k times the units digit of X.

Find the minimum possible value of the sum $X + Y$.

A three-digit number is an integer from 100 to 999; in particular, its hundreds digit must not be 0.

Input

The only line contains an integer k ($2 \le k \le 9$).

Output

Print the minimum possible value of $X + Y$. It can be shown that such numbers X and Y always exist.

Sample Input 1	Sample Output 1
2	537

Sample Explanation

There are two pairs with the minimum sum 537:

- $X = 102$ and $Y = 435$: the six digits 1, 0, 2, 4, 3 and 5 are pairwise distinct, and the hundreds digit of Y is $4 = 2 \times 2$, two times the units digit of X;
- $X = 132$ and $Y = 405$: the six digits 1, 3, 2, 4, 0 and 5 are pairwise distinct, and the hundreds digit of Y is $4 = 2 \times 2$, two times the units digit of X.

This page is intentionally left blank.

Problem F: Funny Chase

This is an interactive problem.

The Fornax Diamond, the most precious gem in the galaxy, has been stolen by Bash, the most famous thief in the galaxy! Alexia, an officer of the Intergalactic Police, is now chasing him. The galaxy is a Cartesian plane, and both Alexia and Bash always stay at points (x, y) with integer coordinates satisfying $|x| \le 10^9$ and $|y| \le 10^9$.

Alexia has a portal gun, which can take her to **any** point of the galaxy. Bash escapes with a portal gun of an older model, which is broken: it always takes him to a **random** point. Due to its low battery, Bash must alternate between a *long jump* and a *short jump*. More precisely, Bash's jumps alternate as follows:

- For the 1-st, 3-rd, 5-th, ... jump (a *long jump*), Bash moves to a point chosen uniformly at random among all points of the galaxy.
- For the 2-nd, 4-th, 6-th, ... jump (a *short jump*), if Bash is at (x, y), he moves to a point (x', y') chosen uniformly at random among all points of the galaxy with $|x - x'| + |y - y'| \le 10^6$.

All random choices are independent, and Bash's jumps do not depend on Alexia's moves.

Fortunately, the Intergalactic Police had hidden a tracking device inside the Fornax Diamond to protect it. Let (x_A, y_A) and (x_B, y_B) be the positions of Alexia and Bash. At the start of the chase, whenever Alexia moves, and whenever Bash makes a long jump, Alexia learns the **square** of the Euclidean distance between them, i.e., $(x_A - x_B)^2 + (y_A - y_B)^2$. However, with its battery low, Bash's old portal gun runs in power-saving mode during a short jump and leaks so much electromagnetic noise that it jams the tracker's signal, so the tracker can only estimate the distance roughly: whenever Bash makes a short jump, the Euclidean distance $\sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$ is first **rounded to the nearest multiple of** $1\,000$ (if the distance is exactly between two multiples of $1\,000$, it is rounded up), then Alexia only learns the **square** of this rounded distance. For example, the distance $\sqrt{4\,020\,041} \approx 2\,005.004$ is rounded to $2\,000$, and Alexia learns $2\,000^2 = 4\,000\,000$.

Initially, Alexia is at $(0, 0)$ and Bash is at a point chosen uniformly at random among all points of the galaxy. Then Alexia and Bash take turns, starting with Alexia:

- Alexia moves to any point of the galaxy (staying in her current position is allowed).
- Bash makes his next jump.

Alexia catches Bash by moving to the point where Bash is. Note that if Bash jumps to Alexia's position, he is not caught yet; he can only be caught in Alexia's next turn if she chooses to stay in her current position. Help Alexia catch Bash using at most $2\,000$ moves.

Interaction Protocol

First, your program reads the number of test cases τ ($1 \le \tau \le 100$). For each test case:

- First, your program reads an integer d – the square of the distance between Alexia's initial position $(0, 0)$ and Bash's initial position.
- Then, your program makes **at most** $2\,000$ moves. For each move:
  - Your program prints two integers x and y ($|x| \le 10^9$; $|y| \le 10^9$), meaning that Alexia moves to (x, y).

– Your program reads an integer d – the square of the distance between Alexia and Bash after Alexia's move. If $d = 0$, Alexia has caught Bash, and your program should continue with the next test case, or terminate if this is the last one.
– Otherwise, Bash jumps, and your program reads an integer d – the square of the distance between Alexia and Bash if this is a long jump, or the square of the distance rounded to the nearest multiple of 1 000 if this is a short jump.

If your program prints an invalid move, or Alexia has not caught Bash after the 2 000-th move, the interactor prints -1 instead of the distance. In this case, your program should terminate immediately to receive a *Wrong Answer* verdict.

After printing each query do not forget to output the end of line and flush[1] the output.

Sample Interaction

Stdin	Stdout	Comment
2		There are two test cases.
		The first test case starts: Alexia is at $(0, 0)$.
25		Bash is at $(3, -4)$. Squared distance $= (0-3)^2 + (0-(-4))^2 = 25$.
	4 3	Alexia moves to $(4, 3)$.
50		Bash is not there. Squared distance $= (4-3)^2 + (3-(-4))^2 = 50$.
170		Bash makes a long jump to $(-7, 10)$. Squared distance $= (4-(-7))^2 + (3-10)^2 = 170$.
	-7 10	Alexia moves to $(-7, 10)$.
0		Alexia catches Bash. Squared distance $= (-7-(-7))^2 + (10-10)^2 = 0$.
		The second test case starts: Alexia is at $(0, 0)$.
4073		Bash is at $(37, -52)$. Squared distance $= (0-37)^2 + (0-(-52))^2 = 4\,073$.
	1 -1	Alexia moves to $(1, -1)$.
3897		Bash is not there. Squared distance $= (1-37)^2 + (-1-(-52))^2 = 3\,897$.
106		Bash makes a long jump to $(6, 8)$. Squared distance $= (1-6)^2 + (-1-8)^2 = 106$.
	2 3	Alexia moves to $(2, 3)$.
41		Bash is not there. Squared distance $= (2-6)^2 + (3-8)^2 = 41$.
4000000		Bash makes a short jump to $(6, 2008)$. Distance $= \sqrt{(2-6)^2 + (3-2008)^2} \approx 2\,005.004$, rounded to $2\,000$. Squared rounded distance $= 2\,000^2 = 4\,000\,000$.
	7 2008	Alexia moves to $(7, 2008)$.
1		Bash is not there. Squared distance $= (7-6)^2 + (2008-2008)^2 = 1$.
3984160		Bash makes a long jump to $(-5, 12)$. Squared distance $= (7-(-5))^2 + (2008-12)^2 = 3\,984\,160$.
	-5 12	Alexia moves to $(-5, 12)$.
0		Alexia catches Bash. Squared distance $= (-5-(-5))^2 + (12-12)^2 = 0$.

The sample interaction only shows how your program interacts with the interactor, and does not show how the answer is deduced.

[1]To flush, use: `fflush(stdout)` or `cout.flush()` in C++; `sys.stdout.flush()` in Python; or see the documentation for other languages.

Problem G: GCD Sequence

For a sequence A of n positive integers $a_1, a_2, \ldots, a_n$, its *GCD sequence* is the sequence B of $n-1$ integers $b_1, b_2, \ldots, b_{n-1}$, where $b_i = \gcd(a_i, a_{i+1})$.

For example, the GCD sequence of $A = [2, 4, 5, 3, 9]$ is $B = [2, 1, 1, 3]$.

Here, $\gcd(x, y)$ denotes the greatest common divisor of integers x and y, which is the largest positive integer that divides both x and y. For example, $\gcd(12, 18) = 6$.

You are given a sequence B of $n-1$ positive integers. Find a sequence A of n positive integers whose GCD sequence is B, such that the sum $a_1 + a_2 + \ldots + a_n$ is minimized, or report that no such sequence exists.

Input

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

The first line of each test case contains an integer n ($2 \le n \le 10^5$) – the length of the sequence A.

The second line contains $n-1$ integers $b_1, b_2, \ldots, b_{n-1}$ ($1 \le b_i \le 10^9$).

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

Output

For each test case, if there is no sequence A whose GCD sequence is B, print -1. Otherwise, print n integers $a_1, a_2, \ldots, a_n$ – a sequence whose GCD sequence is B with the minimum sum.

It can be shown that if such a sequence exists, every a_i in a sequence with the minimum sum does not exceed 10^{18}. If there are multiple solutions, you can output any of them.

Sample Input 1	Sample Output 1
1 5 2 1 1 3	2 2 1 3 3

Sample Explanation

In the first test case, $A = [2, 4, 5, 3, 9]$ has the GCD sequence $[2, 1, 1, 3]$ with sum 23. However, $A = [2, 2, 1, 3, 3]$ has the same GCD sequence with a smaller sum of 11, which is the minimum.

This page is intentionally left blank.

Problem H: House Cleaning Robot

Minh has just bought a house cleaning robot. The floor of Minh's house can be seen as a square grid of size $(2 \cdot 10^9 + 1) \times (2 \cdot 10^9 + 1)$, divided into 1×1 cells. Each cell is denoted by its coordinates (x, y), where x and y are integers with $|x| \leq 10^9$ and $|y| \leq 10^9$. Initially, Minh places the robot at the cell $(0, 0)$, in the middle of the floor.

The robot is pre-programmed with a string s of n characters, each of which is `U`, `D`, `L` or `R`. In the i-th step, the robot moves from its current cell (x, y) as follows:

- If $s_i =$ `U`, it moves to the cell $(x, y + 1)$.
- If $s_i =$ `D`, it moves to the cell $(x, y - 1)$.
- If $s_i =$ `L`, it moves to the cell $(x - 1, y)$.
- If $s_i =$ `R`, it moves to the cell $(x + 1, y)$.

After a test run, Minh feels that the robot does not visit enough distinct cells, so Minh wants to reprogram it. However, the robot is very complicated, so Minh can only insert a string t of at most 10 characters (also consisting of `U`, `D`, `L` and `R`) at some position in the string s.

Finding the optimal way turns out to be too hard, so Minh decides to try some plans instead. There are q plans. In the j-th plan, Minh inserts the string t_j right after the p_j-th character of s (if $p_j = 0$, t_j is inserted **before** the first character of s). For each plan, Minh wants to know the number of distinct cells the robot visits, including the starting cell $(0, 0)$.

Note that the plans are independent: each plan inserts t_j into the original string s.

Input

The first line contains two integers n and q $(1 \leq n, q \leq 10^5)$ – the length of the string s and the number of plans.

The second line contains the string s of n characters, each of which is `U`, `D`, `L` or `R`.

The j-th of the next q lines contains an integer p_j $(0 \leq p_j \leq n)$ and a string t_j $(1 \leq |t_j| \leq 10)$ consisting of characters `U`, `D`, `L` and `R` – the description of the j-th plan.

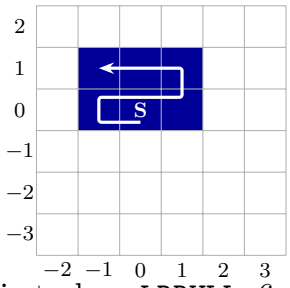
Output

For each plan, print the number of distinct cells the robot visits.

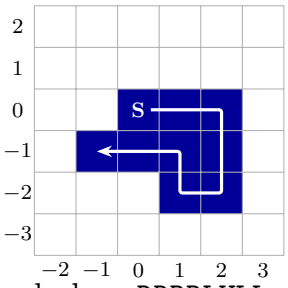
Sample Input 1	Sample Output 1
5 4 RRULL 0 L 2 DDL 3 RD 5 UUU	6 9 6 9

Sample Explanation

The figure below shows the first two plans. The robot starts at the cell marked S, the arrows show its path, and the visited cells are dark blue.



First plan: `LRRULL`, 6 cells



Second plan: `RRDDLULL`, 9 cells

Problem I: I know you don't know; Before you said that, I didn't know, but now I know

Claudia is hosting a logic game for two friends, Alexia and Bash.

The game is played on a grid with R rows and C columns. The rows are numbered from 1 to R from top to bottom, and the columns are numbered from 1 to C from left to right. The cell in row r and column c is denoted by (r, c). Each cell is either **white** or **black**. Claudia, Alexia and Bash all know the grid.

Claudia chooses a **secret** white cell, called the *location*. Claudia tells Alexia only the **row** of the location, and tells Bash only the **column** of the location. Alexia and Bash are perfect logicians who never lie, and all of the above is known to everybody.

Then Alexia and Bash have a conversation. They speak one sentence at a time, in any order. In particular, the same person may say several sentences in a row. Each sentence is one of the following six types, where "you" refers to the other person:

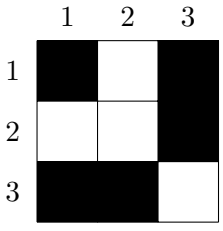
1. I don't know the location.
2. I know the location.
3. I know that you don't know the location.
4. I don't know whether you know the location.
5. Before you said that, I didn't know the location, but I know now.
6. If you said you didn't know the location, I would know the location.

A sentence of type 5 can only be said **immediately after a sentence said by the other person**, and "that" refers only to this last sentence of the other person.

Every sentence refers to what the speaker knows at the moment it is said, taking into account all previous sentences. In a sentence of type 4, the speaker means that among the locations the speaker considers possible, there is at least one where the other person would know the location, and at least one where the other person would not. In a sentence of type 6, the speaker means that if the other person said "I don't know the location" now, then **exactly one** of the locations the speaker considers possible would remain.

A conversation is *valid* if every sentence in it is true, and after the last sentence, both Alexia and Bash know the location, and so does anyone who knows the grid and listens to the conversation.

For example, consider the following grid with 3 rows and 3 columns, whose white cells are $(1, 2)$, $(2, 1)$, $(2, 2)$ and $(3, 3)$.



- If the location is $(1, 2)$, the following conversation is valid:

1. Alexia: "I know that you don't know the location."
2. Bash: "Before you said that, I didn't know the location, but I know now."

Explanation:

- At the beginning of the game, Alexia was told that the secret cell is in row 1, which contains only one white cell $(1, 2)$. Bash was told that the cell is in column 2, which contains two white cells, so Bash does not know the location.
- After Alexia's sentence, Bash knows that the secret cell cannot be $(2, 2)$. This is because, if the secret cell were in row 2, Alexia would consider $(2, 1)$ possible, and if the secret cell were $(2, 1)$, Bash would know the location. So Alexia's sentence would be false. Hence, Bash now knows that the secret cell is $(1, 2)$, and Bash's sentence is true.

- If the location is $(2, 2)$, the following conversation is valid:

1. Alexia: "If you said you didn't know the location, I would know the location."
2. Alexia: "I don't know the location."
3. Bash: "Before you said that, I didn't know the location, but I know now."

Explanation:

- Bash would not know the location only if it is in column 2, i.e., $(1, 2)$ or $(2, 2)$. Each of rows 1 and 2 contains exactly one of these cells, but row 3 contains none. So Alexia's first sentence is true, and it tells everybody that the location is not $(3, 3)$.
- Alexia's second sentence is true only for row 2, so the location is $(2, 1)$ or $(2, 2)$.
- Bash was told column 2. Right before Alexia's second sentence, both $(1, 2)$ and $(2, 2)$ were possible, and now only $(2, 2)$ is. If the location were $(2, 1)$, Bash would have known it all along, so Bash's sentence would be false. Hence, everybody now knows that the location is $(2, 2)$.

- If the location is $(3, 3)$, the following conversation is **not** valid:

1. Alexia: "I know the location."
2. Bash: "I know the location."

Explanation: Both sentences are true, and both Alexia and Bash know the location. However, a listener cannot tell whether the location is $(1, 2)$ or $(3, 3)$.

For each white cell, determine whether there exists a valid conversation with **at most** 200 **sentences** if Claudia chooses that cell as the location, and if so, find one.

Input

Each test contains multiple test cases. The first line contains the number of test cases τ ($1 \le \tau \le 50$). The description of the test cases follows.

The first line of each test case contains two integers R and C ($1 \le R, C \le 7$) – the number of rows and the number of columns of the grid.

Each of the next R lines contains a string of C characters. The c-th character of the r-th line is `W` if the cell (r, c) is white, or `B` if it is black. There is at least one white cell.

It is guaranteed that the sum of $R \cdot C$ over all test cases does not exceed 50.

Output

For the T-th test case, first print a line `Case #`T`:`. Then process the white cells in increasing order of row, and for cells in the same row, in increasing order of column. For each white cell (r, c):

- If there is no valid conversation with at most 200 sentences, print a line with three integers r, c and -1.
- Otherwise, print a line with three integers r, c and k ($0 \le k \le 200$) – the number of sentences in a valid conversation. Then print k lines describing the sentences in order. Each line contains the letter `A` if the sentence is said by Alexia or `B` if it is said by Bash, immediately followed by the type of the sentence (an integer from 1 to 6). For example, `A3` means that Alexia says "I know that you don't know the location".

You do not need to minimize k. If there are multiple valid conversations, you can output any of them.

Sample Input 1	Sample Output 1
1 3 3 BWB WWB BBW	Case #1: 1 2 2 A3 B5 2 1 2 B3 A5 2 2 3 A6 A1 B5 3 3 -1

This page is intentionally left blank.

Problem J: Job Interview

Your company is hiring, and N candidates, numbered from 1 to N, have applied for the same position. Each candidate was interviewed by the same three interviewers, and candidate i received the scores a_i, b_i and c_i from the first, the second and the third interviewer, respectively. No interviewer gave the same score to two different candidates.

You are the assistant of the Head of HR, who is a very meticulous person. Your job is to prepare the candidates' resumes for your boss as a single stack, such that whenever a resume lies directly on top of another one, the candidate of the upper resume is *better* than the candidate of the lower one.

Initially, for two different candidates u and v, candidate u is better than candidate v if and only if u received a higher score than v from **at least two** of the three interviewers. It can be seen that for every pair of different candidates (u, v), exactly one of them is considered better than the other.

Note that being better is not necessarily transitive. For example, suppose that there are three candidates with scores $(1, 2, 3)$, $(2, 3, 1)$ and $(3, 1, 2)$. Then candidate 1 is better than candidate 3 (higher scores from the second and the third interviewers), candidate 3 is better than candidate 2 (from the first and the third interviewers), but candidate 2 is better than candidate 1 (from the first and the second interviewers). Even so, the stack with candidate 1 on top, then candidate 3, then candidate 2 at the bottom is valid: only resumes lying directly on top of each other are compared, so it does not matter that candidate 2 is better than candidate 1.

During the day, your colleagues keep bringing you news, which may change who is better than whom. There are Q events, which happen **one after another**. Each event changes only the comparisons described below, and all other comparisons stay the same. There are four types of events:

- `UP` x: it turns out that candidate x is a relative of the CEO. Candidate x is now better than every other candidate.
- `DOWN` x: it turns out that candidate x is a relative of the CEO's sworn enemy. Every other candidate is now better than candidate x.
- `REV` x y: the hiring committee reviews candidates x and y once more and changes its mind about them. If candidate x was better than candidate y, then candidate y is now better than candidate x, and vice versa.
- `STACK`: nothing changes, but your boss asks you for the stack of all N resumes, according to the current comparisons. Formally, the stack is a permutation $p_1, p_2, \ldots, p_N$ of $1, 2, \ldots, N$, listed from top to bottom, such that for every $1 \le i < N$, candidate p_i is better than candidate p_{i+1} at the moment. Determine whether such a stack exists, and if so, find one.

Note:

- At any moment (at the beginning or after every event), the following statement is always true: for every pair of different candidates (u, v), exactly one of them is considered better than the other.
- At the beginning, the better candidate among a pair (u, v) is determined based on two tuples (a_u, b_u, c_u) and (a_v, b_v, c_v), as described above.
- Let $T_{u,v}^{(i)}$ be the better candidate among the pair (u, v) after the i-th event, $T_{u,v}^{(i)} = T_{u,v}^{(i-1)}$ for every pair of candidates (u, v) and every event i, except for the fact that:
  - If the i-th event is `UP` x, $T_{u,x}^{(i)} = x$ for every candidate $u \neq x$.
  - If the i-th event is `DOWN` x, $T_{u,x}^{(i)} = u$ for every candidate $u \neq x$.
  - If the i-th event is `REV` x y, $T_{x,y}^{(i)} \neq T_{x,y}^{(i-1)}$.
- If the i-th event is `STACK`, you need to find a permutation $p_1, p_2, \ldots, p_N$ of $1, 2, \ldots, N$, satisfying that $T_{p_j,p_{j+1}}^{(i)} = p_j$ for every $1 \leq j < N$, or declare that such a permutation does not exist.

Input

The first line contains two integers N and Q ($1 \leq N \leq 10^5$; $1 \leq Q \leq 2 \cdot 10^5$) – the number of candidates and the number of events.

The i-th of the next N lines contains three integers a_i, b_i and c_i ($1 \leq a_i, b_i, c_i \leq 10^6$) – the scores of candidate i from the first, the second and the third interviewer. The values $a_1, a_2, \ldots, a_N$ are pairwise distinct, and so are $b_1, b_2, \ldots, b_N$ and $c_1, c_2, \ldots, c_N$.

Each of the next Q lines describes an event, in the order they happen, in one of the following formats: `UP` x, `DOWN` x, `REV` x y or `STACK` ($1 \leq x, y \leq N$; $x \neq y$).

There are at most 20 `STACK` **events.**

Output

For each `STACK` event, in the order they appear in the input:

- If there is no valid stack, print a single line `NO`.
- Otherwise, print a line `YES`, followed by a line containing N integers $p_1, p_2, \ldots, p_N$ – a valid stack of resumes, from top to bottom.

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

Sample Input 1	Sample Output 1
3 6 3 1 2 1 3 1 2 2 3 STACK UP 2 REV 3 1 STACK DOWN 2 STACK	YES 3 1 2 YES 2 1 3 YES 1 3 2

Sample Explanation

Initially, candidate 1 is better than candidate 2 (higher scores from the first and the third interviewers), candidate 3 is better than candidate 1 (higher scores from the second and the third interviewers), and candidate 3 is better than candidate 2 (higher scores from the first and the third interviewers).

1. `STACK`: the stack $3, 1, 2$ is valid, since candidate 3 is better than candidate 1, and candidate 1 is better than candidate 2.
2. `UP 2`: candidate 2 is now better than candidates 1 and 3, and candidate 3 is still better than candidate 1.
3. `REV 3 1`: candidate 1 is now better than candidate 3.
4. `STACK`: the stack $2, 1, 3$ is valid.
5. `DOWN 2`: candidates 1 and 3 are now better than candidate 2, and candidate 1 is still better than candidate 3.
6. `STACK`: the stack $1, 3, 2$ is valid.

This page is intentionally left blank.

Problem K: K-Equal Substring

You are given a string S consisting of uppercase English letters. For a positive integer k, a *k-equal substring* of S is a substring of k consecutive characters of S which are all the same letter. Two k-equal substrings are different if they start at different positions. For example, the string `AAAB` has two 2-equal substrings (both are `AA`, starting at positions 1 and 2) and one 3-equal substring.

You may change **at most one** character of S to any uppercase English letter. For each $k = 1, 2, \ldots, |S|$, find the minimum possible number of k-equal substrings of S after the change. Note that the change is chosen independently for each k.

Input

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

Each test case consists of a single line containing the string S ($1 \le |S| \le 10^6$), consisting of uppercase English letters.

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

Output

For each test case, print a line with $|S|$ integers. The k-th integer is the minimum possible number of k-equal substrings of S after changing at most one character.

Sample Input 1	Sample Output 1
2 Z AAB	1 3 0 0

This page is intentionally left blank.

Problem L: Last Hour Resolver

What is the most exciting part of an ICPC event? It is definitely the Last Hour Resolver! During the awarding ceremony, people focus on the screen to watch the scoreboard being unfrozen. Everyone holds their breath, waiting for the result, and screams loudly every time a team successfully solves a problem!

Note:

- *This problem is inspired by the format of ICPC. However, it has been modified and simplified to be suitable for an algorithmic problem. Therefore, the rules stated in this problem are different from the official rules of ICPC.*
- *No knowledge of the official ICPC rules is necessary or advantageous for solving this problem.*
- *You should solve this problem solely based on what is stated explicitly in the problem statement.*

This year, the ICPC Vietnam National Programming Contest attracts n teams, which are numbered from 1 to n. The problemset consists of m problems, which are labeled using the first m uppercase English letters: Problem 1 is labeled as `A`, problem 2 is labeled as `B`, and so on.

An ICPC contest lasts for 5 hours. During the first 4 hours, the scoreboard is available and all contestants can see the results of their opponents. However, the scoreboard is frozen during the last hour, preventing a team from knowing the results of all other teams.

Teams' results are measured by the number of problems solved as well as a value called *penalty*. When a team successfully solves a problem, their *penalty* is increased by a value, which is determined by the moment they make the first correct submission for this problem, and the number of incorrect submissions before this submission. Note that unsolved problems do not affect a team's *penalty*.

Teams are ranked from 1 to n using the following rules:

- Teams with more solved problems are ranked higher.
- Among teams with the same number of solved problems, teams with smaller penalty are ranked higher.
- Among teams with the same number of solved problems and the same penalty, teams with smaller indices are ranked higher.

It can be seen that no two teams can have the same rank.

The frozen scoreboard is an $n \times m$ matrix, where the cell at the i-th row and the j-th column represents the status of the i-th team on the j-th problem, which can be in any of the following forms:

- `+` p: The team solved this problem before the last hour, and their penalty is increased by p.
- `-`: The team did not solve this problem before the last hour, and did not make submissions during the last hour.
- `?` p: The team did not solve this problem before the last hour, and they made

submissions during the last hour. The result is still unknown publicly, and there can be two possible scenarios:

- The cell becomes `+ p`: At least one submission is correct. The team gets one more solved problem and their penalty is increased by p.
- The cell becomes `-`: All submissions are incorrect. The team does not get one more solved problem and their penalty stays the same.

The contest has ended and now the most amazing part begins: Last Hour Resolver! Unfortunately, the resolver program contains a bug: Instead of showing the exact verdicts of submissions during the last hour, the resolver resolves randomly! In particular, every cell in the form `? p` turns into either `+ p` or `-` with equal probability. Obviously, this makes the scoreboard extremely hard to predict!

Since the result is quite random, all the teams would like to know their expected ranking. As they are too tired after 5 hours of coding, they ask for your help. Given the frozen scoreboard, your task is to calculate the expected ranking of all the teams.

Input

The first line contains two integers n and m ($1 \le n \le 2000; 1 \le m \le 13$) – the number of teams and problems, respectively.

In the next n lines, the i-th one contains m strings $s_{i,1}$, $s_{i,2}$, ..., $s_{i,m}$; where $s_{i,j}$ denotes the status of the i-th team on the j-th problem, which is in any of the following forms: "`+` p", "`-`" or "`?` p". p is an integer in range $[1; 2 \cdot 10^4]$.

Output

Print n lines, the i-th one contains an irreducible fraction of the form x/y representing the expected ranking of the i-th team. The fraction x/y is irreducible if x and y are coprime, and $y > 0$. It can be proven that the expected ranking is always a rational number.

Sample Input 1	Sample Output 1
3 3 + 2 + 3 ? 4 - ? 1 + 3 ? 2 - -	5/4 2/1 11/4

Sample Input 2	Sample Output 2
3 1 ? 1 ? 1 ? 1	3/2 2/1 5/2

Sample Explanation

The following image demonstrates the frozen scoreboard in the first sample:

	Solved	Penalty	A	B	C
Team 1	2	5	+2	+3	?4
Team 2	1	3	-	?1	+3
Team 3	0	0	?2	-	-

There are 3 cells of the form "? p", meaning that there are $2^3 = 8$ possible scenarios, with equal probability:

	Solved	Penalty	A	B	C
Team 1	3	9	+2	+3	+4
Team 2	2	4	-	+1	+3
Team 3	1	2	+2	-	-

	Solved	Penalty	A	B	C
Team 1	3	9	+2	+3	+4
Team 2	2	4	-	+1	+3
Team 3	0	0	-	-	-

	Solved	Penalty	A	B	C
Team 1	3	9	+2	+3	+4
Team 3	1	2	+2	-	-
Team 2	1	3	-	-	+3

	Solved	Penalty	A	B	C
Team 1	3	9	+2	+3	+4
Team 2	1	3	-	-	+3
Team 3	0	0	-	-	-

	Solved	Penalty	A	B	C
Team 2	2	4	-	+1	+3
Team 1	2	5	+2	+3	-
Team 3	1	2	+2	-	-

	Solved	Penalty	A	B	C
Team 2	2	4	-	+1	+3
Team 1	2	5	+2	+3	-
Team 3	0	0	-	-	-

	Solved	Penalty	A	B	C
Team 1	2	5	+2	+3	-
Team 3	1	2	+2	-	-
Team 2	1	3	-	-	+3

	Solved	Penalty	A	B	C
Team 1	2	5	+2	+3	-
Team 2	1	3	-	-	+3
Team 3	0	0	-	-	-

- Team 1 has probability $\frac{6}{8}$ of being ranked 1 and probability $\frac{2}{8}$ of being ranked 2. Their expected ranking is $\frac{6}{8} \times 1 + \frac{2}{8} \times 2 = \frac{5}{4}$.
- Team 2 has probability $\frac{2}{8}$ of being ranked 1, probability $\frac{4}{8}$ of being ranked 2, and probability $\frac{2}{8}$ of being ranked 3. Their expected ranking is $\frac{2}{8} \times 1 + \frac{4}{8} \times 2 + \frac{2}{8} \times 3 = 2$.
- Team 3 has probability $\frac{2}{8}$ of being ranked 2 and probability $\frac{6}{8}$ of being ranked 3. Their expected ranking is $\frac{2}{8} \times 2 + \frac{6}{8} \times 3 = \frac{11}{4}$.

This page is intentionally left blank.

Problem M: Monosodium Glutamate

MSG, or *monosodium glutamate*, known colloquially in Vietnam as *bột ngọt*, *mì chính*, or *vị tinh*, is a flavor enhancer commonly used in Vietnamese cuisine. MSG is known for its ability to provide *umami* flavor to soups, broths, and stir-fried dishes. In short, to quote a certain Malaysian uncle, MSG stands for "Make Sh't Good". *Fuiyoh!*

Minh has just opened a restaurant. His menu consists of n dishes, where the i-th dish is of category c_i. One day, a Malaysian uncle came to his restaurant. After tasting each of Minh's n dishes, he shook his head and exclaimed, "Your food taste like sadness. Where the MSG? MSG is the king of flavor. No wonder your restaurant look empty like my marriage. Haiyaaa..." Minh immediately realized this uncle was the real deal. He decided to take his advice and add MSG to his dishes.

After careful analysis, Minh has determined that adding MSG to the i-th dish will increase its tastiness by a_i (a_i can be negative – despite what the uncle might think, MSG does not Make *All* Sh't Good). Furthermore, adding MSG to multiple dishes of the same category will only affect the *last* dish of that category. In other words, if Minh adds MSG to dishes $i_1 < i_2 < \cdots < i_k$ of the same category, only dish i_k will have its tastiness increased by a_{i_k}, while the tastiness of dishes $i_1, i_2, \ldots, i_{k-1}$ will remain unchanged.

Since Minh is too busy to find the optimal way to add MSG to his dishes, he decided to choose a nonempty segment $[l, r]$ of his menu and add MSG to all dishes $l, l+1, \ldots, r$. Minh wants to know the maximum total increase in tastiness he can achieve among all possible segments $[l, r]$. Please help Minh find the answer.

Input

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

- The first line of each test case contains an integer n ($1 \le n \le 10^6$) – the number of dishes in Minh's menu.
- The second line contains n integers $c_1, c_2, \ldots, c_n$ ($1 \le c_i \le n$) – the categories of Minh's dishes.
- The third line contains n integers $a_1, a_2, \ldots, a_n$ ($-10^9 \le a_i \le 10^9$) – the tastiness increases of Minh's dishes.

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

Output

For each test case, print a single integer – the maximum total increase in tastiness.

Sample Input 1	Sample Output 1
4 3 1 1 2 1 -2 3 4 1 2 3 4 5 6 -7 8 2 1 1 -6 -7 5 1 3 2 3 4 100 -1000000000 1 -100 100	3 12 -6 101

Sample Explanation

In the first test case, the optimal segment is $[3, 3]$ with a total increase of 3. Note that for the segment $[1, 3]$, dishes 1 and 2 are both of category 1, so only dish 2 is affected, and the total increase is only $-2 + 3 = 1$.

In the second test case, all dishes are of different categories, so the total increase of a segment is simply the sum of $a_l, a_{l+1}, \ldots, a_r$. The optimal segment is $[1, 4]$ with a total increase of $5 + 6 + (-7) + 8 = 12$.

In the third test case, every segment has a negative total increase. Since the segment must be nonempty, the optimal segment is $[1, 1]$ with a total increase of -6.

In the fourth test case, the optimal segment is $[1, 5]$. Dishes 2 and 4 are both of category 3, so only dish 4 is affected, and the large decrease of dish 2 is ignored. The total increase is $100 + 1 + (-100) + 100 = 101$.