

Problem A. Arranging Teams

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

You are the coach of a competitive programming team with n members. After months of intense training, your team is ready for the “Annual International Coding Pageant” (AICP). In this tournament, teams compete against each other in a series of one-on-one matches.

The tournament structure is peculiar. There are m other teams, and your team will play against each of them. For every match against an opponent team, the players are lined up, and the i -th player from your team competes against the i -th player from the opponent’s team.

Each of your team members has a known strength value. The opponent teams have already submitted their player arrangements, and these are fixed. Your task is to find a single, optimal arrangement for your own team members that will be used against all m opponent teams to maximize your total score across the entire tournament.

You have a team of n members with strengths given by an array of size n . You are competing against m other teams. The k -th opponent team (for k from 1 to m) also has n members, with strengths given by $b_{k,1}, b_{k,2}, \dots, b_{k,n}$.

You need to find a permutation* p of length n . This permutation p will be your team’s final arrangement for all matches. When your team (arranged as p) competes against opponent team k , your player at position i (with strength a_{p_i}) is matched against their player at position i (with strength $b_{k,i}$).

The outcome of an individual competition between a player of strength s_1 and a player of strength s_2 is determined as follows:

- If $s_1 > s_2$, you win and get 1 point.
- If $s_1 = s_2$, it’s a draw, and you get 0.5 points.
- If $s_1 < s_2$, you lose and get 0 points.

Your team’s total score is the sum of points from all individual competitions. That is, the sum of scores for each of your n players against their counterparts in all m opponent teams.

Your goal is to find the arrangement p that maximizes this total score.

Input

The first line contains two integers, n ($1 \leq n \leq 20$) and m ($1 \leq m \leq 5\,000$), the number of members per team and the number of opponent teams.

The second line contains n space-separated integers $a_1, a_2, \dots, a_n$, representing the strengths of your team members ($1 \leq a_i \leq 10^9$).

The next m lines each contain n space-separated integers. The k -th of these lines represents the strengths of the k -th opponent team: $b_{k,1}, b_{k,2}, \dots, b_{k,n}$ ($1 \leq b_{k,i} \leq 10^9$).

Output

Print a single line containing n space-separated integers, representing the optimal arrangement of your team’s strengths that maximizes the total score. If multiple optimal arrangements exist, any one of them will be accepted.

*A permutation of length n is an array consisting of n distinct integers from 1 to n in arbitrary order. For example, $[2, 3, 1, 5, 4]$ is a permutation, but $[1, 2, 2]$ is not a permutation (2 appears twice in the array), and $[1, 3, 4]$ is also not a permutation ($n = 3$ but there is 4 in the array).

Formally, let p be the optimal permutation, you must output $a_{p_1}, a_{p_2}, \dots, a_{p_n}$ separated by whitespace in a single line.

Example

standard input	standard output
2 2 10 20 15 5 12 18	20 10

Note

Your team has strengths $[10, 20]$. There are two opposing teams.

- Opponent 1: $[15, 5]$
- Opponent 2: $[12, 18]$

Let's analyze the positions:

- Position 1: Your player here will face opponents with strengths $[15, 12]$.
- Position 2: Your player here will face opponents with strengths $[5, 18]$.

If you use the arrangement $p = [1, 2]$, the strength of the team is $[10, 20]$:

- Player at Position 1 (strength 10):
 - * vs 15: Lose (0 pts)
 - * vs 12: Lose (0 pts)
 - * Position 1 Score: 0 pts
- Player at Position 2 (strength 20):
 - * vs 5: Win (1 pt)
 - * vs 18: Win (1 pt)
 - * Position 2 Score: 2 pts
- Total Score: $0 + 2 = 2$ pts

If you use the arrangement $p = [2, 1]$, the strength of the team is $[20, 10]$:

- Player at Position 1 (strength 20):
 - * vs 15: Win (1 pt)
 - * vs 12: Win (1 pt)
 - * Position 1 Score: 2 pts
- Player at Position 2 (strength 10):
 - * vs 5: Win (1 pt)
 - * vs 18: Lose (0 pts)
 - * Position 2 Score: 1 pt
- Total Score: $2 + 1 = 3$ pts

The arrangement $[20, 10]$ yields a higher total score and is the optimal answer.

Problem B. Balloon Trip

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

Summer vacation has arrived, and Nobita is once again dreaming of an exciting adventure. Instead of studying for exams or doing homework, he imagines drifting peacefully in a hot-air balloon across the mountains that stretch endlessly beyond his town. Of course, when it comes to turning dreams into reality, there is only one person he can rely on: Doraemon.

The mountains Nobita wants to cross are arranged in a straight line, and each one has a specific height. For convenience, Doraemon numbers the mountains from 1 to n , and Nobita writes down their current heights in a notebook. The height of the mountain i is a_i . At first glance, the plan seems simple: launch the balloon at one mountain, then float to the next, and keep going until the desired destination.

But the balloon journey is not smooth. Each time the balloon moves from one mountain to the next, the balloon must rise or sink to match the new height. The *effort* required is the difference of the two mountain heights. If the next mountain is nearly the same height as the current one, the balloon moves gently. If the difference is large, the balloon struggles and the journey becomes more costly. Nobita quickly realizes that calculating the total effort for a long trip is not easy at all.

Doraemon, seeing Nobita's frustration, pulls out yet another gadget from his pocket. This one has the power to increase the heights of an entire stretch of mountains all at once. With just a flick, Doraemon can make a group of mountains taller. Naturally, Nobita becomes excited because this means he can reshape the mountains before planning his balloon routes. But now the situation is even more complex: the mountain heights are changing, and Nobita still wants to know the cost of his planned trips.

You are given a sequence of operations, describing either Doraemon's gadget use or Nobita's trip queries. There are two types of operations:

1. *Gadget Boost*: Written as **1 1 r x** — Doraemon uses his gadget to adjust every mountain between l and r (inclusive). Each of these mountains has its height increased by x .
2. *Trip Query*: Written as **2 1 r** — Nobita imagines starting at mountain l , then floating to $l+1$, then to $l+2$, and so on until reaching mountain r . The cost of this trip is the sum of the efforts required for each step. Nobita wants to know this total value after all changes so far have been applied.

Your task is to follow this summer adventure, processing every operation in the given order. When Nobita asks about a trip, you must calculate and report the correct cost at that moment.

Input

The first line of input contains two integers n and q ($1 \leq n, q \leq 3 \cdot 10^5$) — the number of mountains and the number of queries.

The second line of input contains n integers $a_1, \dots, a_n$ ($1 \leq a_i \leq 10^9$) — the height of the mountains.

The next q lines contain query descriptions. Each of them contains integers t, l, r and may contain x ($1 \leq t_i \leq 2, 1 \leq l \leq r \leq n, 1 \leq x_i \leq 10^9$). Here $t = 1$ corresponds to the gadget boost and $t = 2$ corresponds to the trip query.

It's guaranteed that the input will contain at least one trip query.

Output

For each trip query, print the answer.

Examples

standard input	standard output
5 6 5 1 4 4 7 2 1 5 1 2 4 3 2 1 5 2 3 3 1 3 5 2 2 2 4	10 4 0 5
10 9 9 3 18 11 17 19 10 7 10 18 1 1 2 10 1 6 8 8 2 2 9 1 9 9 10 1 6 10 15 1 2 7 4 2 1 10 1 1 4 14 2 3 8	45 68 56

Problem C. CK Chang's Shopping Spree

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

One day, Nobita was asking Doraemon for new gadgets. Their friend, CK Chang, came to visit and heard them talking about going shopping at the 22nd Century Department Store.

CK Chang, who is very smart about money, shared his own special rule. “Doraemon,” he said, “I have a rule that helps me buy a lot of things. It’s CK Chang’s 10x Rule: never buy an item unless you have at least 10 times its price in your pocket before you buy it.”

Doraemon thought this was an interesting rule and decided to use it. For example, to buy a “Hopter” (Bamboo Copter) that costs \$30, he must have at least \$300 in his pocket. After buying it, his money would go down by \$30, leaving him with \$270.

Doraemon has some money and a list of cool gadgets he wants to buy for Nobita. To get the most items possible, he needs your help.

Your job is to find out the maximum number of gadgets Doraemon can get if he follows CK Chang’s 10x Rule. You’ll need to figure out the best order for him to buy them in.

Input

The first line contains two integers, n and m ($1 \leq n \leq 2 \cdot 10^5, 1 \leq m \leq 10^{15}$) — n is the number of gadgets available, and m is the money Doraemon has at start.

The second line has n numbers: $p_1, p_2, \dots, p_n$ ($1 \leq p_i \leq 10^9$) — the price of gadgets.

Output

Output a single integer — the maximum number of gadgets Doraemon can buy.

Examples

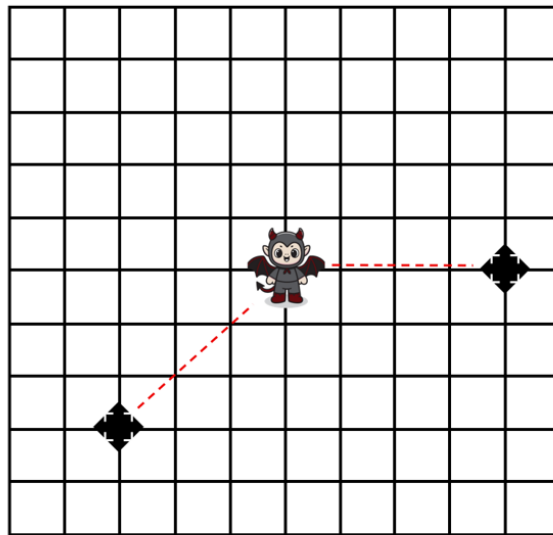
standard input	standard output
5 100 1 2 4 5 8	5
3 100 11 10 12	1

Problem D. Demon on the Grid

Input file: standard input
Output file: standard output
Time limit: 5 seconds
Memory limit: 1024 megabytes

In a vast and mysterious world lies an infinite two-dimensional grid. Each location in this grid is identified by **integer** coordinates (x, y) . You have been informed that **a demon is hiding somewhere in this grid**, and your mission is to find the exact position of the demon before it disappears forever.

You have received n clues from ancient observers who tried to track the demon's movement. Each clue is described by three integers (x_i, y_i, d_i) , meaning that the demon is at a Cartesian distance of $\sqrt{d_i}$ from the point (x_i, y_i) . However, the demon is a master of illusions, so not all clues can be trusted. The demon has created shadow replicas of itself at certain locations, causing some clues to be either **real** or **fake** followed by these conditions: if a clue is real, the demon's actual position must **exactly match** the distance indicated by that clue's coordinates and distance value; if a clue is fake, the demon's position must **not match** that distance.



Unfortunately, you do not know which clues are real and which are fake. To ensure that the demon can be captured, you have to find **the maximum number of real clues** that allows you to locate the demon's position without needing to know in advance which clues are real.

Formally, let c_i represent clue (x_i, y_i, d_i) . Suppose the demon is located at position $o = (o_x, o_y)$, the reality of a clue is defined as follows:

- A clue c_i is real if and only if $(o_x - x_i)^2 + (o_y - y_i)^2 = d_i$.
- A clue c_i is fake if and only if $(o_x - x_i)^2 + (o_y - y_i)^2 \neq d_i$.

You are given a set $S = \{c_1, c_2, \dots, c_n\}$. You need to find the maximum integer k and a position $p \in \mathbb{Z} \times \mathbb{Z}$ such that:

- There is **exactly one** $T \subseteq S$ of size k that makes p a *unique valid solution* for T .
- A position p is a *unique valid solution* for a set T if and only if there is **only** p that is a *valid solution* for T (i.e., p is a *valid solution* for T , and for all $q \in \mathbb{Z} \times \mathbb{Z} - \{p\}$, q is **not** a *valid solution* for T).

- A position p is a *valid solution* for a set T if and only if every clue $c \in T$ is real, and every clue $c' \notin T$ is fake, given that the demon is located at position p .

Input

The first line contains a single integer n ($1 \leq n \leq 10^5$)

The next n lines contain clues. Each contains three integers x_i, y_i, d_i ($-10^5 \leq x_i, y_i \leq 10^5, 1 \leq d_i \leq 10^5$)

It is guaranteed that the demon's coordinates must be integers.

Output

If no solution exists, print -1.

Otherwise, print two lines. On the first line, print the maximum number k of real clues that satisfy the conditions. On the second line, print two space-separated integers representing the demon's position: o_x and o_y .

Examples

standard input	standard output
4 15 0 9 21 0 9 2 0 13 8 0 13	2 18 0
8 0 0 25 0 10 25 5 5 25 20 0 25 20 10 25 25 5 25 10 -10 16 18 -10 16	-1
8 0 0 25 1 0 16 2 0 9 10 0 9 16 0 9 -12 0 25 -11 0 16 -10 0 9	2 13 0

Note

In the first test case:

if $k = 4$: The only possible set includes all four clues, and this set is contradictory.

if $k = 3$: There are four possible sets of three clues, and all of them are contradictory.

if $k = 2$:

- Choosing clues 1 and 2 as real clues uniquely locates the demon's position at $(18, 0)$.
- Choosing clues 3 and 4 leads to two possible positions, $(5, 2)$ and $(5, -2)$, so the demon's location cannot be uniquely determined.

- All other pairs of clues are contradictory.
- Therefore, 1, 2 is the only set of two clues that uniquely determines the demon's position.

if $k = 1$: Choosing any single clue is ambiguous, resulting in multiple solutions.

Therefore, the maximum number of clues that satisfy the conditions is 2, and the corresponding demon position is $(18, 0)$.

In the second test case:

If $k \geq 4$: It is easy to see that there is no way to choose k clues such that they determine a position.

if $k = 3$: There are two possible ways to select three clues that determines a position

- Choosing clue 1, 2, 3 leads to the position $(0, 5)$.
- Choosing clue 4, 5, 6 leads to the position $(20, 5)$.
- Thus, $k = 3$ is invalid because there is more than one possible way to choose k clues that determine a position.

if $k = 2$: There are five possible ways to select two clues that determine a position:

- Choosing clue 1, 3 leads to the position $(5, 0)$.
- Choosing clue 2, 3 leads to the position $(5, 10)$.
- Choosing clue 4, 6 leads to the position $(25, 0)$.
- Choosing clue 5, 6 leads to the position $(25, 10)$.
- Choosing clue 7, 8 leads to the position $(14, -10)$.
- Therefore, $k = 2$ is also invalid.

if $k = 1$: Choosing any single clue is ambiguous, resulting in multiple solutions.

Therefore, the result is -1 .

Problem E. Exotic Array

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

An array $a = [a_1, a_2, \dots, a_n]$ of length n is *exotic* if and only if these two conditions are satisfied.

- All array entries are integers from 1 to m .
- $a_{a_x+y} = x + y$ for all integers x, y such that $1 \leq x, y, a_x + y \leq n$.

You are given two integers n and m . Count the number of exotic arrays a of length n . Since the answer can be very large, print it modulo $10^9 + 7$.

Input

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

Each of the next t lines contains two integers n, m ($1 \leq n, m \leq 10^9$) — the length of the array and the maximum allowed value.

Output

For each test case, print a single integer — the number of exotic arrays a modulo $10^9 + 7$.

Example

standard input	standard output
6	5
1 5	2
2 2	0
2 1	31
3 5	283
5 7	724174902
998244353 1000000000	

Note

In the first test case, $n = 1$. Every array of length 1 is exotic. The answer is 5.

In the second test case, $n = 2, m = 2$. The exotic arrays are $[1, 2]$ and $[2, 2]$.

In the third test case, $n = 2, m = 1$. No exotic arrays exist.

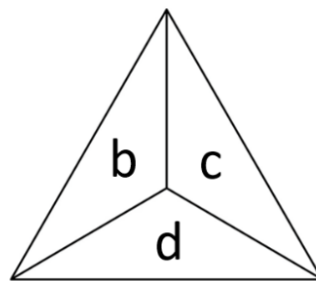
Problem F. Flipping Pyramid

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

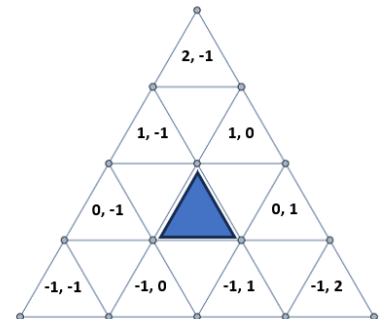
In the ancient land of Tetralia, a magical pyramid roams the infinite two-dimensional triangle grid. This pyramid is a **regular tetrahedron** (a triangular pyramid where all four faces are congruent equilateral triangles). Each of its four faces has a positive integer a, b, c, d marked on it.

Initially, the pyramid stands upright at coordinate $(0,0)$, with one face with integer a resting on the ground. The face with integer b is facing toward the northwest, the face with integer c is facing toward the northeast, and the face with integer d is facing toward the south, as shown in the middle figure.

The infinite two-dimensional triangle grid and its coordinate (x,y) is defined as the right figure, with the blue cell at $(0,0)$ i.e., the starting cell. The $+x$ direction is north, and the $+y$ direction is east.



('a' is at the bottom face)



The pyramid can move by **flipping** itself over one of the three edges of its bottom face; the edge that is used for flipping is called a pivot edge. After a flip, the pyramid lands on the new adjacent cell in the triangle grid, with the new face that contains the pivot edge as one of its sides resting on the ground. The cost of such a move is equal to the number marked on **the new bottom face** (the face touching the ground after the flip).

Your task is to find **the minimum total cost** to move the pyramid to the destination cell (x,y) .

Input

The first line contains 4 integers a, b, c, d ($1 \leq a, b, c, d \leq 10^6$) — four faces of the pyramid.

The second line contains 2 integers x, y ($-10^9 \leq x, y \leq 10^9$) — the coordinates of the destination cell.

Output

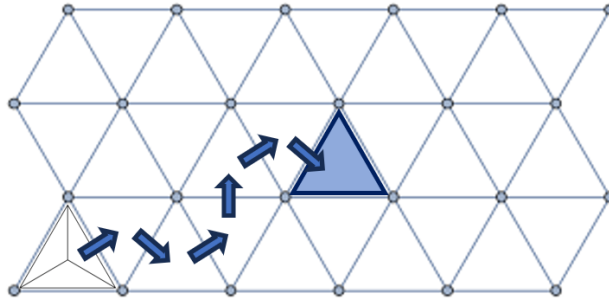
A single integer — the minimum cost to move the pyramid to the destination cell.

Examples

standard input	standard output
1 2 3 4 1 2	15
1 2 3 4 -1 0	6
10 10 10 10 0 0	0

Note

In the first test case, the image below shows one of the moves that results in the minimum cost.



The bottom face of the tetrahedron along the path is $a \rightarrow c \rightarrow d \rightarrow b \rightarrow c \rightarrow a \rightarrow b$, resulting in the total cost of $3 + 4 + 2 + 3 + 1 + 2 = 15$.

Problem G. Graph Game

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

The *Forgotten Caverns* form a network of n chambers, connected by m bidirectional tunnels. This network is a connected, undirected, unweighted graph with vertices numbered from 1 to n .

Each chamber i has a **difficulty** value a_i . Some chambers also contain ancient **treasures**, indicated by $b_i \in \{0, 1\}$: if $b_i = 1$, there is a treasure in chamber i ; if $b_i = 0$, there is none.

You are an adventurer exploring these caverns. You begin in some chamber with magical power p . From your current chamber u , you may move to any adjacent chamber v through a tunnel:

- If $a_v > p$, you must pay $(a_v - p)^2$ coins and set $p \leftarrow a_v$. Otherwise, you pay nothing and p stays the same.
- In both cases, you move to v (so $u \leftarrow v$).

You may visit the same chamber any number of times, and you may use the same tunnel any number of times.

You are asked to process q queries of the following two types:

- **1 u p** — Starting at chamber u with initial power p (guaranteed $p \geq a_u$), compute the minimum number of coins required to visit **all chambers that currently contain treasures at least once**. If there are no treasures, the answer is 0.
- **2 l r** — For all i with $l \leq i \leq r$, toggle whether chamber i contains a treasure: if a treasure exists, remove it; otherwise, place one.

The treasure configuration is **persistent** and updates after each type-2 query.

Input

The first line contains three integers n, m, q ($2 \leq n \leq 10^5$, $n - 1 \leq m \leq 2 \cdot 10^5$, $1 \leq q \leq 10^5$) — the number of chambers, tunnels, and queries respectively.

The second line contains n integers $a_1, a_2, \dots, a_n$ ($1 \leq a_i \leq 10^9$) — the difficulty of each chamber.

The third line contains n integers $b_1, b_2, \dots, b_n$ ($b_i \in \{0, 1\}$) — the initial treasure configuration.

The next m lines each contain two integers x, y ($1 \leq x, y \leq n$, $x \neq y$) — a tunnel between chambers x and y .

The next q lines each describe one query:

- For a query of type-1, the line has the form **1 u p**, where $1 \leq u \leq n$ and $a_u \leq p \leq 10^9$.
- For a query of type-2, the line has the form **2 l r**, where $1 \leq l \leq r \leq n$.

It is guaranteed that the graph is connected, with no multiple edges and no self-loops.

It is guaranteed that there will be at least one type-1 query.

Output

For each query of type-1, output a single integer — the minimum number of coins required.

Example

standard input	standard output
5 7 8	16
6 12 10 15 5	0
1 0 1 0 0	29
2 3	14
4 5	0
1 3	4
3 4	
4 1	
5 3	
2 5	
1 1 6	
1 4 15	
2 3 4	
1 1 6	
1 1 9	
2 2 4	
1 2 20	
1 3 10	

Note

In the first example, the difficulties are $a = [6, 12, 10, 15, 5]$ and the initial treasures are at chambers $\{1, 3\}$.

- Query 1 (1 1 6): Path: $1 \rightarrow 3$. Cost: $(10 - 6)^2 = 16$.
- Query 2 (1 4 15): Path: $4 \rightarrow 5 \rightarrow 3 \rightarrow 1$. Costs: $0 + 0 + 0 = 0$.
- Query 3 (2 3 4): Toggle treasures in $[3, 4]$. Now treasures are at chambers $\{1, 4\}$.
- Query 4 (1 1 6): Path: $1 \rightarrow 3 \rightarrow 2 \rightarrow 5 \rightarrow 4$. Costs: $(10 - 6)^2 + (12 - 10)^2 + (15 - 12)^2 = 29$.
- Query 5 (1 1 9): Path: $1 \rightarrow 3 \rightarrow 2 \rightarrow 5 \rightarrow 4$. Costs: $(10 - 9)^2 + (12 - 10)^2 + (15 - 12)^2 = 14$.
- Query 6 (2 2 4): Toggle treasures in $[2, 4]$. Now treasures are at chambers $\{1, 2, 3\}$.
- Query 7 (1 2 20): Treasures $\{1, 2, 3\}$. Path: $2 \rightarrow 5 \rightarrow 4 \rightarrow 1 \rightarrow 3$. Costs: $0 + 0 + 0 + 0 = 0$.
- Query 8 (1 3 10): Treasures $\{1, 2, 3\}$. Path: $3 \rightarrow 2 \rightarrow 3 \rightarrow 1$. Costs: $0 + (12 - 10)^2 + 0 = 4$.

Problem H. Horse Racing

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

The Royal Bangkok Sports Club is preparing new endurance training for special horses. The racecourse is an nm grid of rectangular fields. Each field (i, j) has a difficulty rating $D_{i,j}$. A higher value means the road there is rougher and requires more effort from the horse. Some fields are soft grass, while others are bumpy and uneven, demanding very high effort. Horses can move between two fields if they **share a common side** (up, down, left, and right).

For training, the committee designs tasks as follows:

- A horse starts from a given field (r, c) .
- It must visit **every field** inside a specified rectangular region $[r_1..r_2] \times [c_1..c_2]$ (including both boundaries) at least once.
- The area of the specified rectangular region is **at most 100 fields**.

The effort of a training route is defined as **the maximum difficulty** among all fields the horse must step on during the route.

The horse may choose any order to visit the fields, and the committee only cares about the minimum possible effort. There are q tasks, and you must report all of them!

Input

The first line contains three integers n, m, q ($1 \leq n, m \leq 500$) and ($1 \leq q \leq 10^5$).

The next n lines each contain m integers, giving the difficulty ratings $D_{i,j}$ of the fields.

- $1 \leq i \leq n$ and $1 \leq j \leq m$
- $0 \leq D_{i,j} \leq 10^9$

Each of the next q lines describes one query with six integers r, c, r_1, c_1, r_2, c_2

- $1 \leq r \leq n$ and $1 \leq c \leq m$
- $1 \leq r_1 \leq r_2 \leq n$
- $1 \leq c_1 \leq c_2 \leq m$
- It is guaranteed that $1 \leq (r_2 - r_1 + 1) \cdot (c_2 - c_1 + 1) \leq 100$

Output

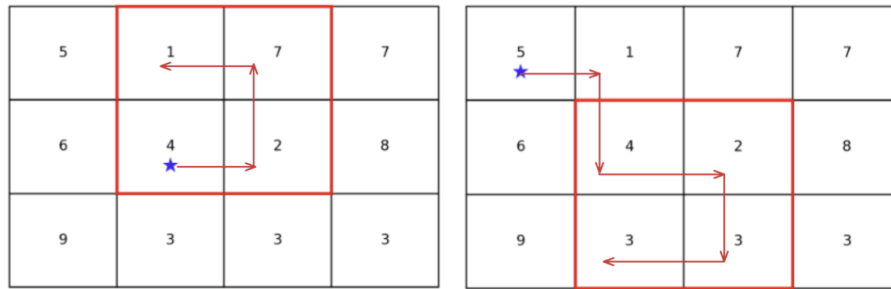
For each query, print a single integer — the minimum effort required for that horse's route.

Example

standard input	standard output
3 4 2 5 1 7 7 6 4 2 8 9 3 3 3 2 2 1 2 2 3 1 1 2 2 3 3	7 5

Note

These are the moves that results in the minimum cost.



Other routes may exist, but whichever path is chosen, the minimum possible effort should be the same.

Problem I. Illuminated Ray Cast

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

There are n vertical poles standing on a straight line at integer positions $x = 1, 2, \dots, n$. The i -th pole has height h_i and is located at $x = i$.

Sunlight comes **from the left** and travels in the direction of the vector $\vec{d} = (t_x, t_y)$ with $t_x > 0$. This means the rays propagate toward increasing x , with vertical slope determined by t_y/t_x .

A pole is **ray cast** (its top is directly lit) if its top lies strictly above the **shadow envelope** formed by all poles to its left.

- The shadow envelope is the upper boundary traced by rays starting from the tops of earlier poles and extending rightward along $\vec{d}$.
- If the top of a pole lies **exactly on** the envelope, then the pole is **not ray cast** (ties are considered blocked).

Your task is to count how many poles are ray cast.

Input

The first line contains t ($1 \leq t \leq 150\,000$) — the number of test cases.

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

The second line contains n integers $h_1, h_2, \dots, h_n$ ($1 \leq h_i \leq 10^9$) — the heights of the poles.

The third line contains two integers t_x and t_y ($1 \leq t_x \leq 10^9$, $-10^9 \leq t_y \leq 10^9$) — the components of the sunlight direction vector $\vec{d}$.

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

Output

For each test case, print a single integer — the number of poles that are ray cast.

Example

standard input	standard output
4	3
5	2
2 3 2 7 4	1
2 1	2
5	
5 7 4 6 3	
2 -1	
4	
5 5 5 5	
3 0	
3	
3 10 7	
1 -2	

Note

Figure 1 corresponds to the first test case with $h = [2, 3, 2, 7, 4]$, $t_x = 2$, $t_y = 1$.

- Poles 1, 2, and 4 are **ray cast**.
- Poles 3 and 5 are **not ray cast** because they lie under the shadow envelope.

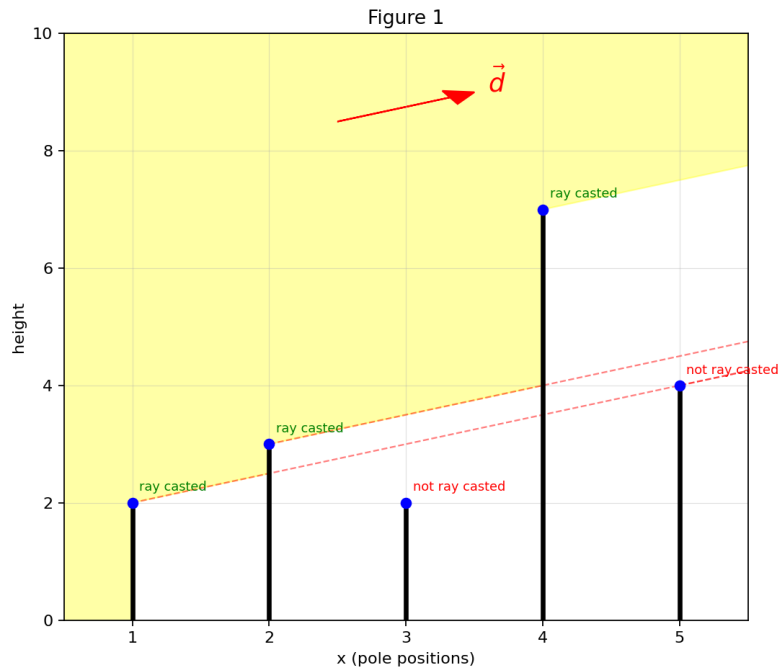
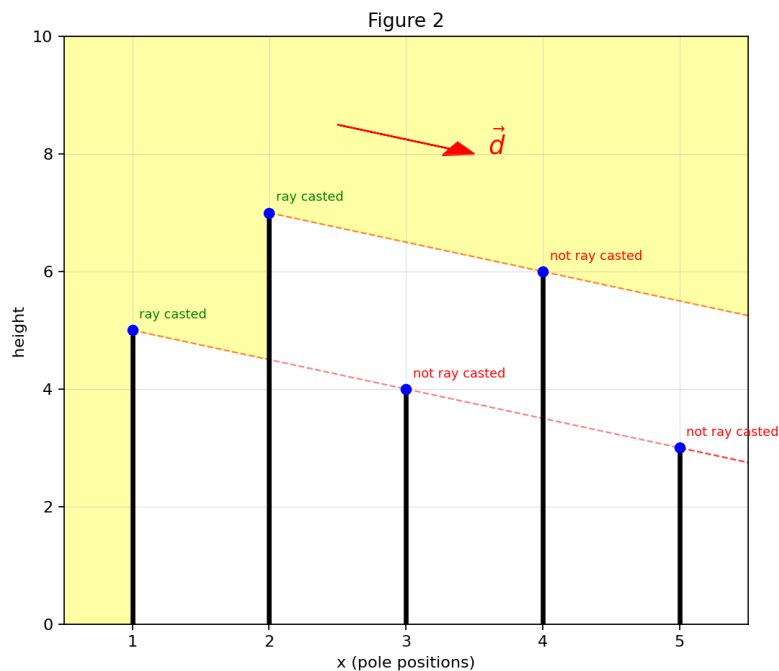


Figure 2 corresponds to the second test case with $h = [5, 7, 4, 6, 3]$, $t_x = 2$, $t_y = -1$.

- Poles 1 and 2 are **ray cast**.
- Poles 3, 4, and 5 are **not ray cast**.
- Notice that pole 4 is **not ray cast even though a ray from pole 1 exactly meets its tip**. By rule, equality with the shadow envelope does not count as lit — ties are considered blocked.



Problem J. Jinglebell

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

It's Christmas Eve! Santa Claus takes a quick trip in his time machine to visit his good friend, Doraemon, in the 22nd century. Seeing Santa so busy, Doraemon offers to help with a futuristic sleigh and some amazing gadgets!

This year, they are delivering two special items:

- Yummy Dorayaki for the good kids (like Nobita and Shizuka).
- A special "Reflection Mirror" gadget for the naughty kids (like Gian) to help them see their mistakes.

But Doraemon has a strict rule from his "Future Gadget Rulebook" to make sure the deliveries are perfect:

- The wonderful, sweet smell of Dorayaki can't be ruined by passing through a naughty kid's house.
- The special power of the Reflection Mirror will get weak if it passes through a good kid's house.

So, for each delivery, Santa must travel from Doraemon's workshop to one kid's house and come straight back to the workshop to pick up the next item. Your job is to help Santa and Doraemon find the shortest total distance they need to travel to deliver everything correctly.

You are given a map with Doraemon's workshop and all the kids' houses. The roads connect these places, and each road has a certain length. Every house is either a "Good Kid" house or a "Bad Kid" house.

Santa has to make one trip for each and every house:

- Start at the workshop.
- Travel to one house to deliver an item.
- Travel all the way back to the workshop.

Here are the two very important rules for the path from the workshop to the house, and from the house back to the workshop:

- When delivering Dorayaki to a Good Kid, the path cannot go through any Bad Kid's house.
- When delivering a Reflection Mirror to a Bad Kid, the path cannot go through any Good Kid's house.

You are guaranteed that Santa can always find a way to reach every house by following these rules.

Input

The first line has one number, n ($1 \leq n \leq 10^5$), which is how many houses there are.

The second line has a string of n letters. Each letter tells you the type of a house. 'G' is for a Good kid's house, and 'B' is for a Bad kid's house.

The third line has one number, m ($1 \leq m \leq 3 \cdot 10^5$), which is how many roads there are on the map.

The next m lines describe the roads. Each line has three numbers: u , v , and d ($0 \leq u, v \leq n, 1 \leq d \leq 10^6$). This means there is a road between place u and place v with a distance of d . Place 0 is Doraemon's workshop, and places 1 to n are the houses.

Output

Print one single number — the shortest total distance Santa must travel for all the deliveries combined.

Example

standard input	standard output
3 GBG 4 0 1 10 0 2 12 1 3 5 2 3 6	74

Problem K. King of Tic-Tac-Toe

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

Nobita is in trouble again! He made a bet with the neighborhood bully, Gian, that he could win a game of Tic-Tac-Toe. Of course, Gian didn't play fair, and the board is now in a very strange state. The number of marks for each player might not be what you'd expect from a normal game. With Gian about to claim victory, Nobita starts to cry for help.

From his pocket, Doraemon pulls out a futuristic gadget: the "Double-Step Stopwatch!" This incredible device can bend time, allowing Nobita a single, golden opportunity: he gets to make two consecutive moves for 'O'. After Nobita uses his special double-move, the game ends immediately. If there are three 'O's in the same row, column, or diagonal of the board, Nobita wins.

You are Nobita's trusted friend, here to help him analyze the board. Doraemon guarantees that the board has at least two empty cells and that neither side has already won.

Given a 3x3 Tic-Tac-Toe board, you must help Nobita determine if it is possible to win after using the Double-Step Stopwatch. If Nobita can make two moves that result in an immediate win, show him the winning board. This means he gets to enjoy a victory feast of dorayaki! Otherwise, report that it is impossible to win.

If multiple winning moves exist, any of them is an acceptable answer.

Input

The input will consist of an integer t ($1 \leq t \leq 1,000$) on the first line, indicating the number of test cases. For each test case, there are 3 lines, each containing 3 characters, representing the Tic-Tac-Toe board.

- 'X' represents a cell occupied by Gian.
- 'O' represents a cell occupied by Nobita.
- '.' represents an empty cell.

Output

- If a win is possible for Nobita ('O'), print the word **YES** on the first line, followed by the 3x3 board state after the two winning moves on the next three lines.
- If a win is not possible, print the single word **NO**.

Example

standard input	standard output
3	YES
X.X	X.X
.O.	OOO
...	...
O.X	YES
.O.	O.X
X..	.O.
X.X	XOO
.X.	NO
...	

Note

In the first test case, Nobita places 'O' at cell (2,1) and (2,3), winning the game on the spot.

In the second test case, Nobita places 'O' at cell (3,2) and (3,3), winning the game on the spot.

Problem L. Label the Tree

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

You are given a rooted tree[†] with n vertices numbered from 1 to n . The root is the vertex 1.

A permutation[‡] $p_1, p_2, \dots, p_n$ is good if and only if, for each $1 \leq i \leq n$, **at least** one of the following conditions is true:

- $p_i = i$.
- Vertex p_i is an ancestor[§] of vertex i .
- Vertex p_i is a descendant[¶] of vertex i .

Count the number of good permutations of length n , modulo 998 244 353.

Input

The first line contains t ($1 \leq t \leq 2\,500$) — the number of test cases.

The first line of each test case contains a single integer n ($2 \leq n \leq 5\,000$) — the number of vertices.

The second line contains $n - 1$ integers $a_2, a_3, \dots, a_n$ ($1 \leq a_i \leq i - 1$) — a_i represents the parent of the vertex i .

It is guaranteed that the sum of n over all test cases does not exceed 5 000.

Output

For each test case, output one number — the number of good permutations modulo 998 244 353.

Example

standard input	standard output
5	3
3	6
1 1	20
3	75
1 2	237554682
6	
1 1 1 2 2	
7	
1 1 2 2 3 3	
13	
1 2 3 4 5 6 7 8 9 10 11 12	

Note

In the first test case, the good permutations are $[1, 2, 3]$, $[2, 1, 3]$, and $[3, 2, 1]$.

In the second test case, all $3! = 6$ permutations are good.

[†]A tree is a connected graph without cycles. A rooted tree is a tree where one vertex is special and called the root.

[‡]A permutation of length n is an array consisting of n distinct integers from 1 to n in arbitrary order. For example, $[2, 3, 1, 5, 4]$ is a permutation, but $[1, 2, 2]$ is not a permutation (2 appears twice in the array), and $[1, 3, 4]$ is also not a permutation ($n = 3$ but there is 4 in the array).

[§]An ancestor of vertex v is any vertex on the simple path from v to the root, including the root, but not including v . The root has no ancestors.

[¶]A descendant of vertex v is any vertex u for which v is an ancestor. No vertex is its own descendant.

Problem M. Mine

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

Nobita was tired of getting bullied by Gian. As he is a cultured citizen, he chose to solve this problem by planting N mines in the school, which is a rectangular field of R rows and C columns. After he finished his business, he went back to his house and told Doraemon what he did. Doraemon was so proud of him, then asked if Nobita remembered where he put the mines. Unsurprisingly, he couldn't recall the location of any mine, and not even the number of mines he put in. Doraemon, however, expected this; he prepared a device that can locate every mine on the field. But the device requires the location of any mine in that field first.

Nobita was given a detector that receives a rectangular area and a **positive integer** k that must be **no greater than 20**. The detector will return the number of mines in that specified area modulo k .

Since Nobita needs to finish this fast for some reason, he can use this device not so many times before it is too late.

Your task is to help Nobita locate **any mine (one mine)** in the field using the detector **no more than 69 times**.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 100$).

For each test case, the first line contains 2 integers R, C ($1 \leq R, C \leq 10^9$) — the number of rows and columns of the field.

The number of mines (i.e. N) in each test case is at least 1 and at most 10^6 . **Note that the number of mines is not provided.**

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

Interaction Protocol

To use the detector, you should print a new line in the following format: “ $? \ k \ r_1 \ c_1 \ r_2 \ c_2$ ” ($1 \leq k \leq 20, 1 \leq r_1 \leq r_2 \leq R, 1 \leq c_1 \leq c_2 \leq C$).

As a response, the detector will return the amount of mines which are located at row i column j in that rectangle ($r_1 \leq i \leq r_2, c_1 \leq j \leq c_2$) modulo k .

To report the answer, you should print a new line in the following format: “ $! \ r \ c$ ” indicating that there is a mine located at row r column c .

If you use the detector more than 69 times, you ask a malformed query, or your answer is wrong, you will get **Wrong answer**.

After printing a query, do not forget to output the end of line and flush the output. To do this, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `System.out.flush()` in Java;
- `flush(output)` in Pascal;
- `stdout.flush()` in Python;

Note that the interactor is **not** adaptive, which means that the hidden arrays do not depend on the queries you make.

Example

standard input	standard output
2	? 3 1 1 3 3
7 6	? 2 5 1 5 6
1	! 2 3
0	? 2 1 5 8 9
10 10	! 4 9
1	

Note

In the first testcase, the field has 7 rows and 6 columns. Suppose that the mines are located in $(2, 3)$, $(5, 3)$, $(5, 6)$.

	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						
7						

1. The first query is the red rectangle. As there is 1 mine and k is 3, $1 \bmod 3 = 1$. The detector returns 1.
2. The second query is the blue rectangle. As there are 2 mines and k is 2, $2 \bmod 2 = 0$. The detector returns 0.
3. The program then reports that there is one mine at $(2, 3)$ which is correct and the program terminates.

In the second testcase, the field has 10 rows and 10 columns. Suppose there is only one mine located in $(4, 9)$.

1. The only query for the mine that stays within row 1 – 8 and column 5 – 9 with $k = 2$. The detector returns 1 as it detects the only mine in the field.
2. The program then reports that there is one mine at $(4, 9)$ which is correct and the program terminates.

Problem N. Nobita's Homework! Help Me Doraemon

Input file: **standard input**
Output file: **standard output**
Time limit: 1.5 seconds
Memory limit: 256 megabytes

Nobita has once again forgotten his homework. This time, Ajarn Nat asked him to design a special grid. Luckily, Doraemon pulls out a grid-maker-from-anywhere gadget that instantly creates such a grid.

The rules of the grid-maker-from-anywhere are as follows:

- Nobita gives Doraemon two sequences a and b of equal length n .
- The first element is always the same, i.e., $a_1 = b_1$.
- Doraemon's gadget uses these to construct a square grid with n rows and n columns.
- Let $f(i, j)$ denote the number on the cell at row i and column j , the value of $f(i, j)$ is defined as follows:
 - $f(1, 1) = a_1 = b_1$
 - $f(i, 1) = a_i$ for all $2 \leq i \leq n$.
 - $f(1, j) = b_j$ for all $2 \leq j \leq n$.
 - $f(i, j) = f(i - 1, j - 1)$ for all $2 \leq i, j \leq n$.

After the grid is built, Ajarn Nat provides Nobita with another sequence x of length n . For each row i of the grid, the gadget aligns x with that row and computes the value $y_i = \sum_{j=1}^n f(i, j) \cdot x_j$

Collecting all y_i for all $1 \leq i \leq n$ forms a new sequence y of length n . Nobita must report this sequence y to finally complete his homework.

Input

The first line contains an integer n ($1 \leq n \leq 10^5$).

The second line contains n integers $a_1, a_2, \dots, a_n$ ($0 \leq a_i < 100$).

The third line contains n integers $b_1, b_2, \dots, b_n$ ($0 \leq b_i < 100$).

The fourth line contains n integers $x_1, x_2, \dots, x_n$ ($0 \leq x_i < 100$).

It is guaranteed that $a_1 = b_1$.

Output

A single line consists of n integers — the sequence y .

Example

standard input	standard output
4 1 3 5 7 1 2 4 8 1 4 1 2	29 17 22 32

Note

From $a = [1, 3, 5, 7]$ and $b = [1, 2, 4, 8]$, the gadget constructs the following grid.

1	2	4	8
3	1	2	4
5	3	1	2
7	5	3	1

The sequence is $x = [1, 4, 1, 2]$.

- The gadget aligns x with the first row, $y_1 = 1 \cdot 1 + 2 \cdot 4 + 4 \cdot 1 + 8 \cdot 2 = 29$.
- The gadget aligns x with the second row, $y_2 = 3 \cdot 1 + 1 \cdot 4 + 2 \cdot 1 + 4 \cdot 2 = 17$.
- The gadget aligns x with the third row, $y_3 = 5 \cdot 1 + 3 \cdot 4 + 1 \cdot 1 + 2 \cdot 2 = 22$.
- The gadget aligns x with the fourth row, $y_4 = 7 \cdot 1 + 5 \cdot 4 + 3 \cdot 1 + 1 \cdot 2 = 32$.