Kotlin Heroes: Episode 4

A. Color Revolution

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

Hmm, how long has it been since the last color revolution? 5 years?! It's totally the time to make a new one!

So the general idea is the following. Division 1 should have n_1 participants. Division 2 should have n_2 and be exactly k times bigger than division 1 ($n_2 = k \cdot n_1$). Division 3 should have $n_3 = k \cdot n_2$ participants. Finally, division 4 should have $n_4 = k \cdot n_3$ participants.

There are n participants on Codeforces in total. So $n_1 + n_2 + n_3 + n_4$ should be exactly equal to n.

You know the values of n and k. **You also know that n and k are chosen in such a way that there exist values n_1, n_2, n_3 and n_4 such that all the conditions are satisfied.**

What will be the number of participants in each division (n_1, n_2, n_3 and n_4) after the revolution?

Input
The first line contains a single integer t ($1 \le t \le 1000$) — the number of testcases.

Each of the next t lines contains two integers n and k ($4 \le n \le 10^9$; $1 \le k \le 500$) — the total number of participants on Codeforces and the size multiplier for the corresponding testcase. In each testcase, n and k are chosen in such a way that the answer exists.

Output
For each testcase print four integers n_1, n_2, n_3 and n_4 such that $n_2 = k \cdot n_1$, $n_3 = k \cdot n_2$, $n_4 = k \cdot n_3$ and $n_1 + n_2 + n_3 + n_4 = n$.

Example

input
4 40 3 1200 7 320802005 400 4 1
output
1 3 9 27 3 21 147 1029 5 2000 800000 320000000 1 1 1 1

B. Boot Camp

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

Berland State University (BSU) is conducting a programming boot camp. The boot camp will last for n days, and the BSU lecturers are planning to give some number of lectures during these days.

Some days of the boot camp are already planned as excursion days, and no lectures should be held during these days. To make sure the participants don't get too tired of learning to program, the number of lectures for each day should not exceed k_1, and the number of lectures for each pair of **consecutive** days should not exceed k_2.

Can you calculate the maximum number of lectures that can be conducted during the boot camp? Formally, find the maximum integer m such that it is possible to choose n non-negative integers c_1, c_2, ..., c_n (where c_i is the number of lectures held during day i) so that:

- $c_1 + c_2 + \cdots + c_n = m$;
- for each excursion day d, $c_d = 0$;
- for each day i, $c_i \le k_1$;
- for each pair of consecutive days $(i, i+1)$, $c_i + c_{i+1} \le k_2$.

Note that there might be some non-excursion days without lectures (i. e., it is possible that $c_i = 0$ even if i is not an excursion day).

Input
The first line contains one integer t ($1 \le t \le 50$) — the number of testcases.

Then the testcases follow, each consists of two lines. The first line contains three integers n, k_1, k_2 ($1 \le n \le 5000$; $1 \le k_1 \le k_2 \le 200\,000$).

The second line contains one string s consisting of exactly n characters, each character is either `0` or `1`. If $s_i = 0$, then day i is an excursion day (so there should be no lectures during that day); if $s_i = 1$, then day i is not an excursion day.

Output
For each test case, print one integer — the maximum possible value of m (the number of lectures that can be conducted).

Example

input
4 4 5 7 1011 4 4 10 0101 5 3 4 11011 6 4 6 011101
output
12 8 8 14

C. Spring Cleaning

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

Tanya wants to organize her bookcase. There are n bookshelves in the bookcase, the i-th bookshelf contains a_i books on it. Tanya will be satisfied if **each bookshelf contains no more than k books**.

Tanya can do one of the two following operations to achieve her goal:

1. Choose exactly one bookshelf and put all the books from it in the storage room (i. e. choose some i and assign $a_i := 0$). During this operation she spends x seconds.
2. Take **all** books from all n bookshelves and distribute them between all n bookshelves *evenly* (the definition of the term is given below). During this operation she spends y seconds.

Consider the sequence a of n **integers**. Then its *even* distribution is such a sequence b of n **integers** that the sum of b equals the sum of a and the value $max(b) - min(b)$ is the minimum possible.

For example, if the array $a = [5, 4, 3]$ then its *even* distribution is $b = [4, 4, 4]$. If $a = [1, 2, 3, 4]$ then its *even* distribution is $b = [2, 3, 3, 2]$ (or any permutation of this array).

Your task is to find the minimum number of seconds Tanya has to spend to obtain the bookcase with no more than k books on **each bookshelf**.

Input
The first line of the input contains one integer t ($1 \le t \le 10^4$) — the number of test cases. Then t test cases follow.

The first line of the test case contains four integers n, k, x and y ($1 \le k \le n \le 2 \cdot 10^5$; $1 \le x, y \le 10^4$) — the number of bookshelves, the maximum required number of books on each bookshelf and the number of seconds Tanya spends during the first and the second operation respectively.

The second line of the test case contains n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le n$), where a_i is the number of books on the i-th bookshelf.

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

Output
For each test case, print the answer — the minimum number of seconds Tanya has to spend to obtain the bookcase with no more than k books on **each bookshelf**.

Example

input
6 5 4 3 5 1 2 2 3 5 5 3 4 5 1 5 1 5 5 5 4 5 6 1 2 5 3 5 4 3 2 10 4 4 1 1

```
4 3 10 2
4 4 1 1
4 1 5 4
1 2 1 3
```

output

```
3
9
6
4
2
9
```

Note

In the first test case of the example, it's optimal to use the first operation on the fifth bookshelf. So the array a becomes $[1, 2, 2, 3, 5] \to [1, 2, 2, 3, 0]$.

In the second test case of the example, it's optimal to use the first operation on the second bookshelf and then use the second operation. So the array a becomes $[1, 5, 1, 5, 5] \to [1, 0, 1, 5, 5] \to [2, 2, 3, 3, 2]$.

In the third test case of the example, it's optimal to use the second operation. So the array a becomes $[1, 2, 5, 3, 5] \to [4, 3, 3, 3, 3]$.

In the fourth test case of the example, it's optimal to use the first operation on the first and the second bookshelves. So the array a becomes $[4, 4, 1, 1] \to [0, 0, 1, 1]$.

In the fifth test case of the example, it's optimal to use the second operation. So the array a becomes $[4, 4, 1, 1] \to [2, 3, 2, 3]$.

D. Constructing the Dungeon

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

Polycarp is developing an RPG game where the main character fights monsters and searches for treasure in dungeons. Now Polycarp is making one of the dungeons the character can explore.

The dungeon consists of n rooms connected by m two-way tunnels, and it is possible to reach every room from every other room using tunnels. The rooms are guarded by monsters (the number of monsters in the i-th room is a_i), and the tunnels contain gold coins (the number of coins in the i-th tunnel is w_i). The i-th two-way tunnel connects rooms v_i and u_i.

Polycarp has already fixed the number of coins in each tunnel (the values of w_i are already known), and now he tries to place the monsters in the rooms (the values of a_i are not known yet). Polycarp wants to choose the number of monsters in each room in such a way that the following two conditions are met:

- the number of coins for the tunnel connecting the rooms x and y should be equal to the minimum of a_x and a_y. That is, for each tunnel i, $w_i = \min(a_{v_i}, a_{u_i})$;
- the number of monsters in the dungeon is as small as possible. That is, the value of $a_1 + a_2 + \cdots + a_n$ is minimum possible.

Help Polycarp to choose the values $a_1, a_2, \ldots, a_n$, or tell him that it is impossible and he has to change something in his dungeon plan.

Input

The first line contains one integer t ($1 \le t \le 100000$) — the number of test cases. Then the test cases follow.

The first line of each test case contains two integers n and m ($2 \le n \le 200000$; $n - 1 \le m \le \min(200000, \frac{n(n-1)}{2})$) — the number of rooms and tunnels in the dungeon, respectively.

Then m lines follow, each line describing one of the tunnels in the dungeon. The i-th line contains three integers v_i, u_i and w_i ($1 \le v_i, u_i \le n$; $v_i \ne u_i$; $1 \le w_i \le 10^9$) denoting a two-way tunnel that connects rooms v_i and u_i, and contains w_i coins. The tunnel system is connected in each test case (it is possible to reach every room from every other room using the tunnels). Each pair of rooms is connected by at most one tunnel.

The sum of n over all test cases does not exceed 200000. Similarly, the sum of m over all test cases does not exceed 200000.

Output

For each test case, print the answer as follows:

If it is impossible to find the values of $a_1, a_2, \ldots, a_n$ satisfying all the constraints, print one single string `NO` on a separate line. Otherwise, print `YES` in the first line, and n integers $a_1, a_2, \ldots, a_n$ in the second line. If there are multiple valid answers, print any of them.

Example

input

```
3
3 2
1 2 1
2 3 1
```

```
5 7
3 2 7
3 4 9
1 5 5
1 2 5
4 1 5
4 2 7
3 1 5
4 4
1 2 5
3 2 2
4 1 3
3 4 4
```

output

```
YES
1 1 1
YES
5 7 9 9 5
NO
```

E. Magic Tricks

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

Masha is going to participate in a talent show conducted by the university she studies at. She wants to impress the audience with lots of different magic tricks!

For one of her tricks, she uses n sponge balls, one of which is a special one. First, she arranges the balls in a row in such a way that the special ball is put on position k (positions are numbered from 1 to n from left to right). After that, she performs m swaps: during the i-th swap, she chooses the ball on position x_i and the ball on position y_i, and swaps them.

Since Masha is a magician, she fakes some of her actions to trick the audience — when she starts performing a swap, she may fake it, so it is not performed (but it looks like it is performed for the audience). There are no constraints on which swaps Masha should fake or should actually perform — for example, she may fake all of the swaps, or even not fake anything at all.

For the trick to work perfectly, the special ball should end up on a specific position — Masha has not decided yet, which position is perfect. Since faking swaps is difficult, for each position she wants to know the minimum number of swaps she has to fake so that the special ball ends up there.

Unfortunately, Masha is a magician, neither a mathematician nor a programmer. So she needs your help in calculating what she wants!

Input

The first line contains three integers n, m and k ($2 \le n \le 2 \cdot 10^5$; $1 \le m \le 2 \cdot 10^5$; $1 \le k \le n$) — the number of balls, the number of swaps and the initial position of the special ball, respectively.

Then m lines follow, the i-th line contains two integers x_i and y_i ($1 \le x_i, y_i \le n$; $x_i \ne y_i$) denoting the i-th swap.

Output

Print n integers. The i-th integer should be the minimum number of swaps Masha has to fake so the special ball ends up on position i (or -1, if Masha cannot put the special ball there).

Examples

input

```
4 5 1
3 4
2 1
4 1
3 1
3 1
```

output

```
2 0 3 1
```

input

```
5 7 4
3 2
3 2
4 2
3 4
4 1
3 2
5 2
```

output

```
2 2 0 3 1
```

input
7 15 5 5 3 4 2 6 1 2 4 1 6 3 7 5 6 4 2 6 4 2 6 6 3 6 3 7 6 2 6 7 2

output
-1 1 1 1 2 1 0

F. Dune II: Battle For Arrakis

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

You're at the last mission in one very old and very popular strategy game Dune II: Battle For Arrakis. The map of the mission can be represented as a rectangular matrix of size $n \times m$. Initially, there are $a_{i,j}$ units of your army in the cell (i, j).

You want to prepare for the final battle, so you want to move all your army into **exactly one cell** of the map (i.e. $nm - 1$ cells should contain 0 units of the army and the remaining cell should contain the entire army).

To do this, you can do some (possibly, zero) number of moves. During one move, you can select **exactly one unit** from some cell and move it to one of the adjacent **by side** cells. I.e. from the cell (i, j) you can move the unit to cells:

- $(i - 1, j)$;
- $(i, j - 1)$;
- $(i + 1, j)$;
- $(i, j + 1)$.

Of course, you want to move all your army into **exactly one cell** as fast as possible. So, you want to know the **minimum** number of moves you need to do that.

And, of course, life goes on, so the situation on the map changes. There are q updates, the i-th update is denoted by three integers x, y, z. This update affects the army in the cell (x, y): after this update, the number of units in the cell (x, y) becomes z (i.e. you replace $a_{x,y}$ with z).

Also, you want to determine, for each i, the **minimum** number of moves needed to move your entire army into **exactly one cell** with the first i updates applied to the initial map. In other words, the map after the i-th update equals the **initial** map with the first i updates applied to it.

Input

The first line of the input contains three integers n, m and q ($1 \le n, m \le 1000; 1 \le q \le 5000$) — the size of the matrix and the number of updates correspondingly.

The next n lines contain m integers each, where the j-th integer in the i-th line is $a_{i,j}$ ($1 \le a_{i,j} \le 10^9$) — the number of units in the cell (i, j).

The next q lines contain three integers each, where the i-th line contains three integers x_i, y_i and z_i ($1 \le x_i \le n; 1 \le y_i \le m; 1 \le z_i \le 10^9$) — the cell in which the number of units updates and the new number of units in this cell correspondingly.

Output

Print $q + 1$ integers $r_0, r_1, r_2, \ldots, r_n$, where r_0 is the **minimum** number of moves you need to move all your army into **exactly one cell**, and r_i for all i from 1 to q is the **minimum** number of moves you need to move all your army into **exactly one cell** after the first i updates.

Examples

input
3 3 1 1 2 3 2 1 2 1 1 2 2 3 100

output
21 22

input
4 4 3 2 5 6 3 4 8 10 5 2 6 7 1 8 4 2 1 1 1 8 2 3 4 4 4 5

output
123 135 129 145

G. Two IP Cameras

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

You have two IP cameras of the same model. Each camera can take photos starting from some moment of time with a fixed period. You can freely choose the starting moment but you can choose the period only as one of k values $p_1, p_2, ..., p_k$ which are chosen by the camera's manufacturer.

You have n moments of interest $x_1, x_2, ..., x_n$. You'd like to configure both cameras in such a way that at least one camera will take a photo in each of these moments. Configuring the camera means setting the moment when it takes the first photo and the gap between two consecutive photos (which should be one of the values $p_1, p_2, ..., p_k$). It's not a problem for you that cameras can take photos at other moments of time — you only care about moments of interest.

Input

The first line contains two integers k and n ($1 \le k \le 10^5$; $2 \le n \le 10^5$) — the number of periods to choose and the number of moments of interest.

The second line contains k integers $p_1, p_2, ..., p_k$ ($1 \le p_1 < p_2 < \cdots < p_k \le 10^6$) — the periods to choose in the ascending order.

The third line contains n integers $x_1, x_2, ..., x_n$ ($1 \le x_1 < x_2 < \cdots < x_n \le 10^6$) — the moments of interest in the ascending order.

Output

Print YES (case insensitive) in the first line if there is a way to configure cameras.

In the second line, print two integers s_1 and cp_1 ($1 \le s_1 \le 10^6$; $1 \le cp_1 \le 10^6$; $cp_1 \in \{p_1, ..., p_k\}$) — the starting moment and the period for the first camera. The period should be one of the given periods.

In the third line, print two integers s_2 and cp_2 ($1 \le s_2 \le 10^6$; $1 \le cp_2 \le 10^6$; $cp_2 \in \{p_1, ..., p_k\}$) — the starting moment and the period for the second camera. The period should be one of the given periods.

If there is no way to configure cameras, print `NO` (case insensitive). If there are multiple ways, you may print any of them.

Examples

input
3 5 3 5 7 1 4 5 7 12

output
YES 1 3 5 7

input
3 2 1 2 3 1 10

output
YES 1 1 10 1

input
3 4 1 2 3 5 7 9 11

output
YES 5 1

5 1

input
3 4 10 20 100 2 3 4 7
output
NO

H. Game with Segments

time limit per test: 3 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

Alice and Bob are playing a game (yet again).

They have two sequences of segments of the coordinate axis: a sequence of n *initial segments*: $[l_1, r_1]$, $[l_2, r_2]$, ..., $[l_n, r_n]$, and a sequence of m *terminal segments*: $[L_1, R_1]$, $[L_2, R_2]$, ..., $[L_m, R_m]$. At the beginning of the game, they choose one of the *initial segments* and set it as the *current segment*.

Alice and Bob make alternating moves: Alice makes the first move, Bob makes the second move, Alice makes the third one, and so on. During each move, the current player **must** shrink the *current segment* either by increasing its left endpoint by 1, or by decreasing its right endpoint by 1. So, if the current segment is $[c_l, c_r]$, it becomes either $[c_l + 1, c_r]$, or $[c_l, c_r - 1]$.

If at the beginning of the game or after Bob's move the *current segment* coincides with one of the *terminal segments*, Bob wins. If the *current segment* becomes degenerate ($c_l = c_r$), and Bob hasn't won yet, Alice wins. If the *current segment* coincides with one of the *terminal segments* after Alice's move, nothing happens — the game continues.

Both players play optimally — if they can win, they always use a strategy that leads them to victory in the minimum number of turns, and if they cannot win, they try to prolong the game, using the strategy allowing them to make the maximum possible number of moves until their defeat.

For each of the *initial segments* you have to determine who will win the game if this segment is chosen as the *current segment* at the beginning of the game. If Bob wins, you also have to calculate the number of moves Alice will make before her defeat.

Input

The first line contains two integers n and m ($1 \le n, m \le 2 \cdot 10^5$) — the number of *initial segments* and *terminal segments*, respectively.

Then n lines follow, the i-th line contains two integers l_i and r_i ($1 \le l_i < r_i \le 10^6$) — the endpoints of the i-th *initial segment*.

Then m lines follow, the i-th line contains two integers L_i and R_i ($1 \le L_i < R_i \le 10^6$) — the endpoints of the i-th *terminal segment*.

Note that some of the segments given in the input may coincide.

Output

Print n integers, the i-th of them should describe the result of the game if the i-th *initial segment* is chosen at the beginning of the game:

- if Alice wins, print -1;
- if Bob wins, print the number of moves Alice will make before she is defeated.

Examples

input
1 1 4 7 4 7
output
0

input
1 2 2 5 2 4 3 5
output
-1

input
2 1 1 5

```
1 4
2 3
```

output

```
-1 1
```

I. Pac-Man 2.0

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

Polycarp is developing a new version of an old video game "Pac-Man". Though he really enjoyed playing the original game, he didn't like some aspects of it, so he decided to alter the rules a bit.

In Polycarp's version, you play as Pac-Man and you have to collect pellets scattered over the game world while avoiding dangerous ghosts (no difference from the original yet). Polycarp didn't like the fact that there was no escape from the ghosts in the original, so, in his version, the game world is divided into n safe zones with m one-directional pathways between them — and it is guaranteed that Pac-Man can reach any safe zone from any other. Since safe zones are safe, the ghosts cannot attack Pac-Man while it is there, it is in danger only while traversing the pathways. Pac-Man starts the game in the safe zone s.

All pellets are scattered over the safe zones; initially, the i-th safe zone contains a_i pellets (and if Pac-Man is in a safe zone, it may freely collect all the pellets in it). The pellets disappear after being collected, but after the last pellet in the game world is collected, new pellets spawn in the safe zones in the same quantity as before (a_i new pellets spawn in the i-th zone). The pellets can be respawned any number of times, so the game is essentially infinite.

Polycarp has already determined the structure of the game world and the number of pellets in each safe zone. Now he is trying to find out if the game is difficult enough. There are q goals in the game, the i-th goal is to collect at least C_i pellets from the beginning of the game. Polycarp denotes the *difficulty* of the i-th goal as the minimum number of times the player has to traverse a one-directional pathway in order to collect C_i pellets (since only traversing a pathway puts Pac-Man in danger). If some pathway is traversed multiple times while Pac-Man is collecting the pellets, it is included in the answer the same number of times.

Help Polycarp to calculate the difficulty of each goal!

Input

The first line contains four integers n, m, q and s ($2 \le n \le 15$; $n \le m \le n(n-1)$; $1 \le q \le 5000$; $1 \le s \le n$) — the number of safe zones, the number of pathways, the number of goals and the index of the starting safe zone, respectively.

The second line contains n integers a_1, a_2, ..., a_n ($1 \le a_i \le 10^9$), where a_i is the initial number of pellets in the i-th safe zone (and the number of pellets that spawn in the i-th safe zone when the last pellet in the world is collected).

Then m lines follow, each line contains two integers v_i and u_i ($1 \le v_i, u_i \le n$; $v_i \ne u_i$) denoting a one-directional pathway from the safe zone v_i to the safe zone u_i. Each ordered pair (v_i, u_i) appears in this section at most once (there are no multiple pathways from v_i to u_i), and it is possible to reach every safe zone from every other safe zone using these pathways.

The last line contains q integers C_1, C_2, ..., C_q ($1 \le C_i \le 10^{15}$), where C_i is the minimum number of pellets the player has to collect in order to fulfil the i-th goal.

Output

For each goal i, print one integer — its difficulty (the minimum number of times the player has to traverse along some pathway in order to collect at least C_i pellets).

Examples

input

```
3 4 2 1
3 1 2
1 2
2 1
1 3
3 1
5 8
```

output

```
1
3
```

input

```
5 7 4 2
1 3 2 2 1
2 3
4 2
3 4
3 1
1 4
5 4
4 5
7 14 23 27
```

output

```
2
6
10
13
```

input

```
4 4 3 3
2 3 1 4
3 4
4 1
1 2
2 3
13 42 1337
```

output

```
3
13
401
```

Note

Consider the first test. In order to collect 5 pellets, the player should collect 3 pellets in the safe zone 1 (which is starting), move to zone 3, collect 2 pellets there.

In order to collect 8 pellets, the player should collect 3 pellets in the safe zone 1, go to 2, collect 1 pellet, go to 1 without getting pellets, go to 3, collect 2 pellets. Now the last pellet in the world is collected, so they are respawned. The player can collect 2 pellets in the safe zone 3 and now the number of collected pellets is 8.

Consider the second test.

In order to collect 7 pellets let's do the following: $2(+3) \rightarrow 3(+2) \rightarrow 4(+2)$. In such a way 7 pellets were collected.

In order to collect 14 pellets let's do the following: $2(+3) \rightarrow 3(+2) \rightarrow 1(+1) \rightarrow 4(+2) \rightarrow 5(+1)$ `respawn of pellets` $5(+1) \rightarrow 4(+2) \rightarrow 2(+3)$. In such a way 15 pellets were collected.

In order to collect 23 pellets let's do the following: $2(+3) \rightarrow 3(+2) \rightarrow 1(+1) \rightarrow 4(+2) \rightarrow 5(+1)$ `respawn of pellets` $5(+1) \rightarrow 4(+2) \rightarrow 2(+3) \rightarrow 3(+2) \rightarrow 1(+1)$ `respawn of pellets` $1(+1) \rightarrow 4(+2) \rightarrow 2(+3)$. In such a way 24 pellets were collected.