A. Journey Planning

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

Tanya wants to go on a journey across the cities of Berland. There are n cities situated along the main railroad line of Berland, and these cities are numbered from 1 to n.

Tanya plans her journey as follows. First of all, she will choose some city c_1 to start her journey. She will visit it, and after that go to some other city $c_2 > c_1$, then to some other city $c_3 > c_2$, and so on, until she chooses to end her journey in some city $c_k > c_{k-1}$. So, the sequence of visited cities $[c_1, c_2, \dots, c_k]$ should be strictly increasing.

There are some additional constraints on the sequence of cities Tanya visits. Each city i has a beauty value b_i associated with it. If there is only one city in Tanya's journey, these beauty values imply no additional constraints. But if there are multiple cities in the sequence, then for any pair of adjacent cities c_i and c_{i+1}, the condition $c_{i+1} - c_i = b_{c_{i+1}} - b_{c_i}$ must hold.

For example, if $n = 8$ and $b = [3, 4, 4, 6, 6, 7, 8, 9]$, there are several three possible ways to plan a journey:

- $c = [1, 2, 4]$;
- $c = [3, 5, 6, 8]$;
- $c = [7]$ (a journey consisting of one city is also valid).

There are some additional ways to plan a journey that are not listed above.

Tanya wants her journey to be as beautiful as possible. The beauty value of the whole journey is the sum of beauty values over all visited cities. Can you help her to choose the optimal plan, that is, to maximize the beauty value of the journey?

Input

The first line contains one integer n ($1 \le n \le 2 \cdot 10^5$) — the number of cities in Berland.

The second line contains n integers b_1, b_2, ..., b_n ($1 \le b_i \le 4 \cdot 10^5$), where b_i is the beauty value of the i-th city.

Output

Print one integer — the maximum beauty of a journey Tanya can choose.

Examples

input
```
6
10 7 1 9 10 15
```
output
```
26
```

input
```
1
400000
```
output
```
400000
```

input
```
7
8 9 26 11 12 29 14
```
output
```
55
```

Note

The optimal journey plan in the first example is $c = [2, 4, 5]$.

The optimal journey plan in the second example is $c = [1]$.

The optimal journey plan in the third example is $c = [3, 6]$.

B. Navigation System

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

The map of Bertown can be represented as a set of n intersections, numbered from 1 to n and connected by m one-way roads. It is possible to move along the roads from any intersection to any other intersection. The length of some path from one intersection to another is the number of roads that one has to traverse along the path. The shortest path from one intersection v to another intersection u is the path that starts in v, ends in u and has the minimum length among all such paths.

Polycarp lives near the intersection s and works in a building near the intersection t. Every day he gets from s to t by car. Today he has chosen the following path to his workplace: p_1, p_2, ..., p_k, where $p_1 = s$, $p_k = t$, and all other elements of this sequence are the intermediate intersections, listed in the order Polycarp arrived at them. Polycarp never arrived at the same intersection twice, so all elements of this sequence are pairwise distinct. **Note that you know Polycarp's path beforehand (it is fixed), and it is not necessarily one of the shortest paths from s to t.**

Polycarp's car has a complex navigation system installed in it. Let's describe how it works. When Polycarp starts his journey at the intersection s, the system chooses some shortest path from s to t and shows it to Polycarp. Let's denote the next intersection in the chosen path as v. If Polycarp chooses to drive along the road from s to v, then the navigator shows him the same shortest path (obviously, starting from v as soon as he arrives at this intersection). However, if Polycarp chooses to drive to another intersection w instead, the navigator **rebuilds** the path: as soon as Polycarp arrives at w, the navigation system chooses some shortest path from w to t and shows it to Polycarp. The same process continues until Polycarp arrives at t: if Polycarp moves along the road recommended by the system, it maintains the shortest path it has already built; but if Polycarp chooses some other path, the system **rebuilds** the path by the same rules.

Here is an example. Suppose the map of Bertown looks as follows, and Polycarp drives along the path $[1, 2, 3, 4]$ ($s = 1$, $t = 4$):

Check the picture by the link http://tk.codeforces.com/a.png

1. When Polycarp starts at 1, the system chooses some shortest path from 1 to 4. There is only one such path, it is $[1, 5, 4]$;
2. Polycarp chooses to drive to 2, which is not along the path chosen by the system. When Polycarp arrives at 2, the navigator **rebuilds** the path by choosing some shortest path from 2 to 4, for example, $[2, 6, 4]$ (note that it could choose $[2, 3, 4]$);
3. Polycarp chooses to drive to 3, which is not along the path chosen by the system. When Polycarp arrives at 3, the navigator **rebuilds** the path by choosing the only shortest path from 3 to 4, which is $[3, 4]$;
4. Polycarp arrives at 4 along the road chosen by the navigator, so the system does not have to rebuild anything.

Overall, we get 2 rebuilds in this scenario. Note that if the system chose $[2, 3, 4]$ instead of $[2, 6, 4]$ during the second step, there would be only 1 rebuild (since Polycarp goes along the path, so the system maintains the path $[3, 4]$ during the third step).

The example shows us that the number of rebuilds can differ even if the map of Bertown and the path chosen by Polycarp stays the same. Given this information (the map and Polycarp's path), can you determine the minimum and the maximum number of rebuilds that could have happened during the journey?

Input

The first line contains two integers n and m ($2 \le n \le m \le 2 \cdot 10^5$) — the number of intersections and one-way roads in Bertown, respectively.

Then m lines follow, each describing a road. Each line contains two integers u and v ($1 \le u, v \le n$, $u \ne v$) denoting a road from intersection u to intersection v. All roads in Bertown are pairwise distinct, which means that each ordered pair (u, v) appears at most once in these m lines (but if there is a road (u, v), the road (v, u) can also appear).

The following line contains one integer k ($2 \le k \le n$) — the number of intersections in Polycarp's path from home to his workplace.

The last line contains k integers p_1, p_2, ..., p_k ($1 \le p_i \le n$, all these integers are pairwise distinct) — the intersections along Polycarp's path in the order he arrived at them. p_1 is the intersection where Polycarp lives ($s = p_1$), and p_k is the intersection where Polycarp's workplace is situated ($t = p_k$). It is guaranteed that for every $i \in [1, k - 1]$ the road from p_i to p_{i+1} exists, so the path goes along the roads of Bertown.

Output

Print two integers: the minimum and the maximum number of **rebuilds** that could have happened during the journey.

Examples

input

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

output

```
1 2
```

input

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

output

```
0 0
```

input

```
8 13
8 7
8 6
7 5
7 4
6 5
6 4
5 3
5 2
4 3
4 2
3 1
2 1
1 8
5
8 7 5 2 1
```

output

```
0 3
```

C. World of Darkraft: Battle for Azathoth

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

Roma is playing a new expansion for his favorite game World of Darkraft. He made a new character and is going for his first grind.

Roma has a choice to buy **exactly one** of n different weapons and **exactly one** of m different armor sets. Weapon i has attack modifier a_i and is worth ca_i coins, and armor set j has defense modifier b_j and is worth cb_j coins.

After choosing his equipment Roma can proceed to defeat some monsters. There are p monsters he can try to defeat. Monster k has defense x_k, attack y_k and possesses z_k coins. Roma can defeat a monster if his weapon's attack modifier is larger than the monster's defense, and his armor set's defense modifier is larger than the monster's attack. That is, a monster k can be defeated with a weapon i and an armor set j if $a_i > x_k$ and $b_j > y_k$. After defeating the monster, Roma takes all the coins from them. During the grind, Roma can defeat as many monsters as he likes. Monsters do not respawn, thus each monster can be defeated at most one.

Thanks to Roma's excessive donations, we can assume that he has an infinite amount of in-game currency and can afford any of the weapons and armor sets. Still, he wants to maximize the profit of the grind. The profit is defined as the total coins obtained from all defeated monsters minus the cost of his equipment. Note that Roma **must** purchase a weapon and an armor set even if he can not cover their cost with obtained coins.

Help Roma find the maximum profit of the grind.

Input

The first line contains three integers n, m, and p ($1 \le n, m, p \le 2 \cdot 10^5$) — the number of available weapons, armor sets and monsters respectively.

The following n lines describe available weapons. The i-th of these lines contains two integers a_i and ca_i ($1 \le a_i \le 10^6$, $1 \le ca_i \le 10^9$) — the attack modifier and the cost of the weapon i.

The following m lines describe available armor sets. The j-th of these lines contains two integers b_j and cb_j ($1 \le b_j \le 10^6$, $1 \le cb_j \le 10^9$) — the defense modifier and the cost of the armor set j.

The following p lines describe monsters. The k-th of these lines contains three integers x_k, y_k, z_k ($1 \le x_k, y_k \le 10^6$, $1 \le z_k \le 10^3$) — defense, attack and the number of coins of the monster k.

Output

Print a single integer — the maximum profit of the grind.

Example

input
2 3 3 2 3 4 7 2 4 3 2 5 11 1 2 4 2 1 6 3 4 6
output
1

D. Reachable Strings

time limit per test: 3 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

In this problem, we will deal with binary strings. Each character of a binary string is either a `0` or a `1`. We will also deal with substrings; recall that a substring is a contiguous subsequence of a string. We denote the substring of string s starting from the l-th character and ending with the r-th character as $s[l \dots r]$. The characters of each string are numbered from 1.

We can perform several operations on the strings we consider. Each operation is to choose a substring of our string and replace it with another string. There are two possible types of operations: replace `011` with `110`, or replace `110` with `011`. For example, if we apply exactly one operation to the string `110011110`, it can be transformed into `011011110`, `110110110`, or `110011011`.

Binary string a is considered *reachable* from binary string b if there exists a sequence s_1, s_2, ..., s_k such that $s_1 = a$, $s_k = b$, and for every $i \in [1, k-1]$, s_i can be transformed into s_{i+1} using exactly one operation. Note that k can be equal to 1, i. e., **every string is reachable from itself**.

You are given a string t and q queries to it. Each query consists of three integers l_1, l_2 and len. To answer each query, you have to determine whether $t[l_1 \dots l_1 + len - 1]$ is reachable from $t[l_2 \dots l_2 + len - 1]$.

Input

The first line contains one integer n ($1 \le n \le 2 \cdot 10^5$) — the length of string t.

The second line contains one string t ($|t| = n$). Each character of t is either `0` or `1`.

The third line contains one integer q ($1 \le q \le 2 \cdot 10^5$) — the number of queries.

Then q lines follow, each line represents a query. The i-th line contains three integers l_1, l_2 and len ($1 \le l_1, l_2 \le |t|$, $1 \le len \le |t| - \max(l_1, l_2) + 1$) for the i-th query.

Output

For each query, print either YES if $t[l_1 \dots l_1 + len - 1]$ is reachable from $t[l_2 \dots l_2 + len - 1]$, or `NO` otherwise. You may print each letter in any register.

Example

input
5 11011 3 1 3 3 1 4 2 1 2 3
output
Yes Yes No

E. Treeland and Viruses

time limit per test: 3 seconds

memory limit per test: 512 megabytes

input: standard input

output: standard output

There are n cities in Treeland connected with $n-1$ bidirectional roads in such that a way that any city is reachable from any other; in other words, the graph of cities and roads is a tree. Treeland is preparing for a seasonal virus epidemic, and currently, they are trying to evaluate different infection scenarios.

In each scenario, several cities are initially infected with different virus species. Suppose that there are k_i virus species in the i-th scenario. Let us denote v_j the initial city for the virus j, and s_j the propagation speed of the virus j. The spread of the viruses happens in turns: first virus 1 spreads, followed by virus 2, and so on. After virus k_i spreads, the process starts again from virus 1.

A spread turn of virus j proceeds as follows. For each city x not infected with any virus at the start of the turn, at the end of the turn it becomes infected with virus j if and only if there is such a city y that:

- city y was infected with virus j at the start of the turn;
- the path between cities x and y contains at most s_j edges;
- all cities on the path between cities x and y (excluding y) were uninfected with any virus at the start of the turn.

Once a city is infected with a virus, it stays infected indefinitely and can not be infected with any other virus. The spread stops once all cities are infected.

You need to process q independent scenarios. Each scenario is described by k_i virus species and m_i important cities. For each important city determine which the virus it will be infected by in the end.

Input

The first line contains a single integer n ($1 \le n \le 2 \cdot 10^5$) — the number of cities in Treeland.

The following $n - 1$ lines describe the roads. The i-th of these lines contains two integers x_i and y_i ($1 \le x_i, y_i \le n$) — indices of cities connecting by the i-th road. It is guaranteed that the given graph of cities and roads is a tree.

The next line contains a single integer q ($1 \le q \le 2 \cdot 10^5$) — the number of infection scenarios. q scenario descriptions follow.

The description of the i-th scenario starts with a line containing two integers k_i and m_i ($1 \le k_i, m_i \le n$) — the number of virus species and the number of important cities in this scenario respectively. It is guaranteed that $\sum_{i=1}^{q} k_i$ and $\sum_{i=1}^{q} m_i$ do not exceed $2 \cdot 10^5$.

The following k_i lines describe the virus species. The j-th of these lines contains two integers v_j and s_j ($1 \le v_j \le n$, $1 \le s_j \le 10^6$) – the initial city and the propagation speed of the virus species j. It is guaranteed that the initial cities of all virus species within a scenario are distinct.

The following line contains m_i distinct integers $u_1, \ldots, u_{m_i}$ ($1 \le u_j \le n$) — indices of important cities.

Output

Print q lines. The i-th line should contain m_i integers — indices of virus species that cities $u_1, \ldots, u_{m_i}$ are infected with at the end of the i-th scenario.

Example

input

```
7
1 2
1 3
2 4
2 5
3 6
3 7
3
2 2
4 1
7 1
1 3
2 2
4 3
7 1
1 3
3 3
1 1
4 100
7 100
1 2 3
```

output

```
1 2
1 1
1 1 1
```

F. Blocks and Sensors

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

Polycarp plays a well-known computer game (we won't mention its name). Every object in this game consists of three-dimensional blocks — axis-aligned cubes of size $1 \times 1 \times 1$. These blocks are unaffected by gravity, so they can float in the air without support. The blocks are placed in cells of size $1 \times 1 \times 1$; each cell either contains exactly one block or is empty. Each cell is represented by its coordinates (x, y, z) (the cell with these coordinates is a cube with opposite corners in (x, y, z) and $(x + 1, y + 1, z + 1)$) and its contents $a_{x,y,z}$; if the cell is empty, then $a_{x,y,z} = 0$, otherwise $a_{x,y,z}$ is equal to the type of the block placed in it (the types are integers from 1 to $2 \cdot 10^5$).

Polycarp has built a large structure consisting of blocks. This structure can be enclosed in an axis-aligned rectangular parallelepiped of size $n \times m \times k$, containing all cells (x, y, z) such that $x \in [1, n]$, $y \in [1, m]$, and $z \in [1, k]$. After that, Polycarp has installed $2nm + 2nk + 2mk$ sensors around this parallelepiped. A sensor is a special block that sends a ray in some direction and shows the type of the first block that was hit by this ray (except for other sensors). The sensors installed by Polycarp are adjacent to the borders of the parallelepiped, and the rays sent by them are parallel to one of the coordinate axes and directed inside the parallelepiped. More formally, the sensors can be divided into 6 types:

- there are mk sensors of the first type; each such sensor is installed in $(0, y, z)$, where $y \in [1, m]$ and $z \in [1, k]$, and it sends a ray that is parallel to the Ox axis and has the same direction;
- there are mk sensors of the second type; each such sensor is installed in $(n + 1, y, z)$, where $y \in [1, m]$ and $z \in [1, k]$, and it sends a ray that is parallel to the Ox axis and has the opposite direction;
- there are nk sensors of the third type; each such sensor is installed in $(x, 0, z)$, where $x \in [1, n]$ and $z \in [1, k]$, and it sends a ray that is parallel to the Oy axis and has the same direction;
- there are nk sensors of the fourth type; each such sensor is installed in $(x, m + 1, z)$, where $x \in [1, n]$ and $z \in [1, k]$, and it sends a ray that is parallel to the Oy axis and has the opposite direction;
- there are nm sensors of the fifth type; each such sensor is installed in $(x, y, 0)$, where $x \in [1, n]$ and $y \in [1, m]$, and it sends a ray that is parallel to the Oz axis and has the same direction;
- finally, there are nm sensors of the sixth type; each such sensor is installed in $(x, y, k + 1)$, where $x \in [1, n]$ and $y \in [1, m]$, and it sends a ray that is parallel to the Oz axis and has the opposite direction.

Polycarp has invited his friend Monocarp to play with him. Of course, as soon as Monocarp saw a large parallelepiped bounded by sensor blocks, he began to wonder what was inside of it. Polycarp didn't want to tell Monocarp the exact shape of the figure, so he provided Monocarp with the data from all sensors and told him to try guessing the contents of the parallelepiped by himself.

After some hours of thinking, Monocarp has no clue about what's inside the sensor-bounded space. But he does not want to give up, so he decided to ask for help. Can you write a program that will analyze the sensor data and construct any figure that is consistent with it?

Input

The first line contains three integers n, m and k ($1 \le n, m, k \le 2 \cdot 10^5$, $nmk \le 2 \cdot 10^5$) — the dimensions of the parallelepiped.

Then the sensor data follows. For each sensor, its data is either 0, if the ray emitted from it reaches the opposite sensor (there are no blocks in between), or an integer from 1 to $2 \cdot 10^5$ denoting the type of the first block hit by the ray. The data is divided into 6 sections (one for each type of sensors), each consecutive pair of sections is separated by a blank line, and the first section is separated by a blank line from the first line of the input.

The first section consists of m lines containing k integers each. The j-th integer in the i-th line is the data from the sensor installed in $(0, i, j)$.

The second section consists of m lines containing k integers each. The j-th integer in the i-th line is the data from the sensor installed in $(n + 1, i, j)$.

The third section consists of n lines containing k integers each. The j-th integer in the i-th line is the data from the sensor installed in $(i, 0, j)$.

The fourth section consists of n lines containing k integers each. The j-th integer in the i-th line is the data from the sensor installed in $(i, m + 1, j)$.

The fifth section consists of n lines containing m integers each. The j-th integer in the i-th line is the data from the sensor installed in $(i, j, 0)$.

Finally, the sixth section consists of n lines containing m integers each. The j-th integer in the i-th line is the data from the sensor installed in $(i, j, k + 1)$.

Output

If the information from the input is inconsistent, print one integer -1.

Otherwise, print the figure inside the parallelepiped as follows. The output should consist of nmk integers: $a_{1,1,1}$, $a_{1,1,2}$, ..., $a_{1,1,k}$, $a_{1,2,1}$, ..., $a_{1,2,k}$, ..., $a_{1,m,k}$, $a_{2,1,1}$, ..., $a_{n,m,k}$, where $a_{i,j,k}$ is the type of the block in (i, j, k), or 0 if there is no block there. If there are multiple figures consistent with sensor data, describe any of them.

For your convenience, the sample output is formatted as follows: there are n separate sections for blocks having $x = 1$, $x = 2$, ..., $x = n$; each section consists of m lines containing k integers each. **Note that this type of output is acceptable, but you may print the integers with any other formatting instead (even all integers on the same line), only their order matters.**

Examples

input

```
4 3 2

1 4
3 2
6 5

1 4
3 2
6 7
```

```
1 4
1 4
0 0
0 7

6 5
6 5
0 0
0 7

1 3 6
1 3 6
0 0 0
0 0 7

4 3 5
4 2 5
0 0 0
0 0 7
```

output

```
1 4
3 0
6 5

1 4
3 2
6 5

0 0
0 0
0 0

0 0
0 0
0 7
```

input

```
1 1 1

0

0

0

0

0

0
```

output

```
0
```

input

```
1 1 1

0

0

1337

0

0

0
```

output

```
-1
```

input

```
1 1 1

1337

1337

1337

1337

1337

1337
```

output

```
1337
```