Codeforces Round #625 (Div. 2, based on Technocup 2020 Final Round)

A. Contest for Robots

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

Polycarp is preparing the first programming contest for robots. There are n problems in it, and a lot of robots are going to participate in it. Each robot solving the problem i gets p_i points, and the score of each robot in the competition is calculated as the sum of p_i over all problems i solved by it. For each problem, p_i is an integer not less than 1.

Two corporations specializing in problem-solving robot manufacturing, "Robo-Coder Inc." and "BionicSolver Industries", are going to register two robots (one for each corporation) for participation as well. Polycarp knows the advantages and flaws of robots produced by these companies, so, for each problem, he knows precisely whether each robot will solve it during the competition. Knowing this, he can try predicting the results — or manipulating them.

For some reason (which absolutely cannot involve bribing), Polycarp wants the "Robo-Coder Inc." robot to outperform the "BionicSolver Industries" robot in the competition. Polycarp wants to set the values of p_i in such a way that the "Robo-Coder Inc." robot gets **strictly more** points than the "BionicSolver Industries" robot. However, if the values of p_i will be large, it may look very suspicious — so Polycarp wants to minimize the maximum value of p_i over all problems. Can you help Polycarp to determine the minimum possible upper bound on the number of points given for solving the problems?

Input

The first line contains one integer n ($1 \le n \le 100$) — the number of problems.

The second line contains n integers r_1, r_2, ..., r_n ($0 \le r_i \le 1$). $r_i = 1$ means that the "Robo-Coder Inc." robot will solve the i-th problem, $r_i = 0$ means that it won't solve the i-th problem.

The third line contains n integers b_1, b_2, ..., b_n ($0 \le b_i \le 1$). $b_i = 1$ means that the "BionicSolver Industries" robot will solve the i-th problem, $b_i = 0$ means that it won't solve the i-th problem.

Output

If "Robo-Coder Inc." robot cannot outperform the "BionicSolver Industries" robot by any means, print one integer -1.

Otherwise, print the minimum possible value of $\max\limits_{i=1}^{n} p_i$, if all values of p_i are set in such a way that the "Robo-Coder Inc." robot gets **strictly more** points than the "BionicSolver Industries" robot.

Examples

input
5 1 1 1 0 0 0 1 1 1 1
output
3

input
3 0 0 0 0 0 0
output
-1

input
4 1 1 1 1 1 1 1 1
output
-1

input
9 1 0 0 0 0 0 0 0 1 0 1 1 0 1 1 1 1 0
output

4

Note

In the first example, one of the valid score assignments is $p = [3, 1, 3, 1, 1]$. Then the "Robo-Coder" gets 7 points, the "BionicSolver" — 6 points.

In the second example, both robots get 0 points, and the score distribution does not matter.

In the third example, both robots solve all problems, so their points are equal.

B. 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 $[c_1, c_2, \ldots, c_k]$ 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 $c = [3, 6]$.

C. Remove Adjacent

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

You are given a string s consisting of lowercase Latin letters. Let the length of s be $|s|$. You may perform several operations on this string.

In one operation, you can choose some index i and **remove** the i-th character of s (s_i) if **at least one** of its adjacent characters is the *previous* letter in the Latin alphabet for s_i. For example, the *previous* letter for b is a, the *previous* letter for s is r, the letter a has no *previous* letters. Note that after each removal the length of the string decreases by one. So, the index i should satisfy the condition $1 \le i \le |s|$ during each operation.

For the character s_i adjacent characters are s_{i-1} and s_{i+1}. The first and the last characters of s both have only one adjacent character (unless $|s| = 1$).

Consider the following example. Let $s =$ bacabcab.

1. During the first move, you can remove the first character $s_1 =$ b because $s_2 =$ a. Then the string becomes $s =$ acabcab.
2. During the second move, you can remove the fifth character $s_5 =$ c because $s_4 =$ b. Then the string becomes $s =$ acabab.
3. During the third move, you can remove the sixth character $s_6 =$'b' because $s_5 =$ a. Then the string becomes $s =$ acaba.
4. During the fourth move, the only character you can remove is $s_4 =$ b, because $s_3 =$ a (or $s_5 =$ a). The string becomes $s =$ acaa and you cannot do anything with it.

Your task is to find the maximum possible number of characters you can remove if you choose the sequence of operations optimally.

Input

The first line of the input contains one integer $|s|$ ($1 \le |s| \le 100$) — the length of s.

The second line of the input contains one string s consisting of $|s|$ lowercase Latin letters.

Output

Print one integer — the maximum possible number of characters you can remove if you choose the sequence of moves optimally.

Examples

input
8 bacabcab
output
4

input
4 bcda
output
3

input
6 abbbbb
output
5

Note

The first example is described in the problem statement. Note that the sequence of moves provided in the statement is not the only, but it can be shown that the maximum possible answer to this test is 4.

In the second example, you can remove all but one character of s. The only possible answer follows.

1. During the first move, remove the third character $s_3 =$ d, s becomes bca.
2. During the second move, remove the second character $s_2 =$ c, s becomes ba.
3. And during the third move, remove the first character $s_1 =$ b, s becomes a.

D. 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
```

E. 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

F. 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