Problem I
Interval Product

It's normal to feel worried and tense the day before a programming contest. To relax, you went out for a drink with some friends in a nearby pub. To keep your mind sharp for the next day, you decided to play the following game. To start, your friends will give you a sequence of N integers $X_1, X_2, \ldots, X_N$. Then, there will be K rounds; at each round, your friends will issue a command, which can be:

- a *change* command, when your friends want to change one of the values in the sequence; or
- a *product* command, when your friends give you two values I, J and ask you if the product $X_I \times X_{I+1} \times \ldots \times X_{J-1} \times X_J$ is positive, negative or zero.

Since you are at a pub, it was decided that the penalty for a wrong answer is to drink a pint of beer. You are worried this could affect you negatively at the next day's contest, and you don't want to check if Ballmer's peak theory is correct. Fortunately, your friends gave you the right to use your notebook. Since you trust more your coding skills than your math, you decided to write a program to help you in the game.

Input

Each test case is described using several lines. The first line contains two integers N and K, indicating respectively the number of elements in the sequence and the number of rounds of the game ($1 \leq N, K \leq 10^5$). The second line contains N integers X_i that represent the initial values of the sequence ($-100 \leq X_i \leq 100$ for $i = 1, 2, \ldots, N$). Each of the next K lines describes a command and starts with an uppercase letter that is either "`C`" or "`P`". If the letter is "`C`", the line describes a *change* command, and the letter is followed by two integers I and V indicating that X_I must receive the value V ($1 \leq I \leq N$ and $-100 \leq V \leq 100$). If the letter is "`P`", the line describes a *product* command, and the letter is followed by two integers I and J indicating that the product from X_I to X_J, inclusive must be calculated ($1 \leq I \leq J \leq N$). Within each test case there is at least one *product* command.

Output

For each test case output a line with a string representing the result of all the *product* commands in the test case. The i-th character of the string represents the result of the i-th *product* command. If the result of the command is positive the character must be "`+`" (plus); if the result is negative the character must be "`-`" (minus); if the result is zero the character must be "`0`" (zero).

Sample input	**Output for the sample input**

`4 6`
`-2 6 0 -1`
`C 1 10`
`P 1 4`
`C 3 7`
`P 2 2`
`C 4 -5`
`P 1 4`
`5 9`
`1 5 -2 4 3`
`P 1 2`
`P 1 5`
`C 4 -5`
`P 1 5`
`P 4 5`
`C 3 0`
`P 1 5`
`C 4 -5`
`C 4 -5`
	`0+-`
	`+-+-0`

F
Jumping monkey

You are a hunter chasing a monkey in the forest, trying to shoot it down with your all-powerful automatic machine gun. The monkey is hiding somewhere behind the branches of one of the trees, out of your sight. You can aim at one of the trees and shoot; your bullets are capable of going through the branches and killing the monkey instantly if it happens to be in that tree. If it isn't, the monkey takes advantage of the time it takes you to reload and takes a leap into a neighbouring tree without you noticing. It never stays in the same place after a shot. You would like to find out whether there is an strategy that allows you to capture the monkey for sure, irrespective of its initial location and subsequent jumps. If so, you need to determine the shortest sequence of shots guaranteeing this.

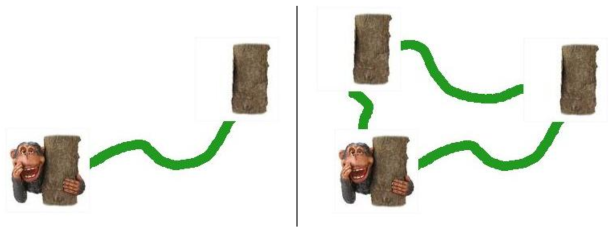

Figure 2

As an example, consider the situation in which there are only two neighboring trees in the forest (left hand side of Figure 2). It is then possible to make sure you capture the monkey by shooting twice at the same tree. Your first shot succeeds if the monkey happened to be there in the first place. Otherwise, the monkey was behind the other tree and it will necessarily have moved when you shoot for the second time.

However, depending on the shape of the forest it may not possible for you to ensure victory. One example of this is if there are three trees, all connected to one another (right hand side of Figure 2). No matter where you aim at, there are always two possible locations for the monkey at any given moment. (Note that here we are concerned with the worst-case scenario where the monkey may consistently guess your next target tree).

Input

The input consists of several test cases, separated by single blank lines. Each test case begins with a line containing two integers n and m ($1 \leq n \leq 21$); n is the number of trees in the forest, and m is the number of adjacency relations between trees. Each of the following m lines contains two distinct integers between 0 and $n-1$ (inclusive), the identifiers of the trees in an adjacent pair. The order of both trees within a pair carries no meaning, and no pair appears more than once. You may further assume that no tree is adjacent to itself, and there is always a path between any two trees in the forest.

The test cases will finish with a line containing only two zeros (also preceded with a blank line).

Output

Print a line for each test case. The line should contain the single word `Impossible` if the task is impossible. Otherwise, it must contain the shortest sequence of shots with the required property, in the format L: $V_1 V_2 \ldots V_L$, where L is the length of the sequence, and $V_1, V_2, \ldots, V_L$ are space-separated integers containing the identifiers of the trees to shoot at in the right order. If several shortest sequences exist, print the lexicographically smallest one. (A sequence is smaller than another in lexicographic order if the first element on which they differ is smaller in the first one).

Sample Input

```
2 1
0 1

3 3
0 1
1 2
2 0

4 3
0 1
2 3
1 3

0 0
```

Sample Output

```
2: 0 0
Impossible
4: 1 3 3 1
```

Joining Couples - ICPC Latin America Regional 2012

Air traffic regulations in Nlogonia require that each city must register exactly one outbound flight to another city. Passengers can use this flight only in the direction registered, that is, there may be a flight registered from city X to city Y and no flight registered from city Y to city X. Thus, the number of registered flights is equal to the number of cities. This rule, as one can imagine, makes air travel somewhat complicated, but tradition and a strong ruling by the Queen makes any changes difficult. Besides, some companies even make a profit from the problems caused by the rule.

The Association for Couple Matching (ACM) is setting up a new service to help customers find their long lasting soulmates: the Internet Connecting Program for Couples (ICPC). The service consists of computing the minimum total number of flights a couple needs to take to meet one another (perhaps in a city where neither of them lives in). Assuming the couple's starting cities are A and B, the agency will try to find a city C such that C is reachable by air travel from both A and B, and the sum of the number of flights needed to go from A to C and the number of flights needed to go from B to C is minimized. Note that C may be equal to A or B or both.

You will be given the list of all available flights, and a list of queries consisting of pairs of cities where the members of a couple live. For each query, you must compute the minimum total number of flights that are needed for them to meet.

Input

Each test case is described using several lines. The first line contains an integer N representing the number of cities ($2 \leq N \leq 10^5$). Cities are identified by different integers from 1 to N. The second line contains N integers F_i, where F_i indicates that the registered outbound flight from city i is to city F_i ($1 \leq F_i \leq N$, $F_i = i$ for $i = 1, 2, \ldots, N$). The third line contains an integer Q representing the number of queries ($1 \leq Q \leq 10^5$). Each of the next Q lines describes a query with two integers A and B indicating the couple's starting cities ($1 \leq A, B \leq N$). Within each test case, if it is possible to travel by air from city X to city Y, the maximum number of flights needed to do so is 10^4.

Output

For each test case output Q lines. In the i-th line write an integer with the answer to the i-th query. If the corresponding couple can meet by air travel, write the minimum total number of flights that the couple must take to meet one another; if it is impossible for the couple to meet by air travel, write the number '`-1`'.

Sample Input

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

Sample Output

```
1
2
0
-1
3
3
-1
1
```

719 Glass Beads

Once upon a time there was a famous actress. As you may expect, she played mostly Antique Comedies most of all. All the people loved her. But she was not interested in the crowds. Her big hobby were beads of any kind. Many bead makers were working for her and they manufactured new necklaces and bracelets every day. One day she called her main *Inspector of Bead Makers* (*IBM*) and told him she wanted a very long and special necklace.

The necklace should be made of glass beads of different sizes connected to each other but without any thread running through the beads, so that means the beads can be disconnected at any point. The actress chose the succession of beads she wants to have and the IBM promised to make the necklace. But then he realized a problem. The joint between two neighbouring beads is not very robust so it is possible that the necklace will get torn by its own weight. The situation becomes even worse when the necklace is disjoined. Moreover, the point of disconnection is very important. If there are small beads at the beginning, the possibility of tearing is much higher than if there were large beads. IBM wants to test the robustness of a necklace so he needs a program that will be able to determine the worst possible point of disjoining the beads.

The description of the necklace is a string $A = a_1a_2 \ldots a_m$ specifying sizes of the particular beads, where the last character a_m is considered to precede character a_1 in circular fashion.

The disjoint point i is said to be worse than the disjoint point j if and only if the string $a_ia_{i+1} \ldots a_na_1 \ldots a_{i-1}$ is lexicografically smaller than the string $a_ja_{j+1} \ldots a_na_1 \ldots a_{j-1}$. String $a_1a_2 \ldots a_n$ is lexicografically smaller than the string $b_1b_2 \ldots b_n$ if and only if there exists an integer $i, i \leq n$, so that $a_j = b_j$, for each $j, 1 \leq j < i$ and $a_i < b_i$.

Input

The input consists of N cases. The first line of the input contains only positive integer N. Then follow the cases. Each case consists of exactly one line containing necklace description. Maximal length of each description is 10000 characters. Each bead is represented by a lower-case character of the english alphabet (`a`–`z`), where $a < b \ldots < z$.

Output

For each case, print exactly one line containing only one integer – number of the bead which is the first at the worst possible disjoining, i.e. such i, that the string $A[i]$ is lexicographically smallest among all the n possible disjoinings of a necklace. If there are more than one solution, print the one with the lowest i.

Sample Input

```
4
helloworld
amandamanda
dontcallmebfu
aaabaaa
```

Sample Output

```
10
11
6
5
```

Problem J
Jupiter Attacks!

Problem code name: `jupiter`

Jupiter is invading! Major cities have been destroyed by Jovian spacecrafts and humanity is fighting back. Nlogonia is spearheading the counter-offensive, by hacking into the spacecrafts' control system.

Unlike Earthling computers, in which usually a byte has 2^8 possible values, Jovian computers use bytes with B possible values, $\{0, 1, \ldots, B-1\}$. Nlogonian software engineers have reverse-engineered the firmware for the Jovian spacecrafts, and plan to sabotage it so that the ships eventually self-destruct.

As a security measure, however, the Jovian spacecrafts run a supervisory program that periodically checks the integrity of the firmware, by hashing portions of it and comparing the result against known good values. To hash the portion of the firmware from the byte at position i to the byte at position j, the supervisor uses the hash function

$$H(f_i, \ldots f_j) = \sum_{k=0}^{j-i} B^k f_{j-k} \pmod{P}$$

where P is a prime number. For instance, if $B = 20$ and $P = 139$, while bytes 2 to 5 of the firmware have the values $f_2 = 14$, $f_3 = 2$, $f_4 = 2$, and $f_5 = 4$, then

$$\begin{aligned} H(f_2, \ldots f_5) &= B^0 f_5 + B^1 f_4 + B^2 f_3 + B^3 f_2 \pmod{P} \\ &= 20^0 \times 4 + 20^1 \times 2 + 20^2 \times 2 + 20^3 \times 14 \pmod{139} \\ &= 4 + 40 + 800 + 112000 \pmod{139} \\ &= 112844 \pmod{139} \\ &= 115 \end{aligned}$$

The Nlogonian cryptologists need to find a way to sabotage the firmware without tripping the supervisor. As a first step, you have been assigned to write a program to simulate the interleaving of two types of commands: editing bytes of the firmware by the Nlogonian software engineers, and computing hashes of portions of the firmware by the Jovian supervisory program. At the beginning of the simulation the value of every byte in the firmware is zero.

Input

Each test case is described using several lines. The first line contains four integers B, P, L and N, where B is the number of possible values of a Jovian byte, P is the modulus of the Jovian hash ($2 \le B < P \le 10^9$ and P prime), L is the length (number of Jovian bytes) of the spacecrafts' firmware, and N is the number of commands to simulate ($1 \le L, N \le 10^5$). At the beginning of the simulation the value of every byte in the firmware is $f_i = 0$ for $1 \le i \le L$. Each of the next N lines describes a command to simulate. Each command description starts with an uppercase letter that is either '`E`' or '`H`', with the following meanings.

'`E`' $\rightarrow$ The line describes an *edit command*. The letter is followed by two integers I and V indicating that the byte at position I of the firmware (that is, f_I) must receive the value V ($1 \le I \le L$ and $0 \le V \le B-1$).

'`H`' $\rightarrow$ The line describes a *hash command*. The letter is followed by two integers I and J indicating that $H(f_I, \ldots f_J)$ must be computed ($1 \le I \le J \le L$).

The last test case is followed by a line containing four zeros.

Output

For each test case output the results of the hash commands in the input. In the i-th line write an integer representing the result of the i-th hash command. Print a line containing a single character '-' (hyphen) after each test case.

Sample input

```
20 139 5 7
E 1 12
E 2 14
E 3 2
E 4 2
E 5 4
H 2 5
E 2 14
10 1000003 6 11
E 1 3
E 2 4
E 3 5
E 4 6
E 5 7
E 6 8
H 1 6
E 3 0
E 3 9
H 1 3
H 4 6
999999935 999999937 100000 7
E 100000 6
E 1 7
H 1 100000
E 50000 8
H 1 100000
H 25000 75000
H 23987 23987
0 0 0 0
```

Output for the sample input

```
115
-
345678
349
678
-
824973478
236724326
450867806
0
-
```

F - Factors

The fundamental theorem of arithmetic states that every integer greater than 1 can be uniquely represented as a product of one or more primes. While unique, several arrangements of the prime factors may be possible. For example:

```
10 = 2 * 5                  20 = 2 * 2 * 5
   = 5 * 2                     = 2 * 5 * 2
                               = 5 * 2 * 2
```

Let $f(k)$ be the number of different arrangements of the prime factors of k. So $f(10) = 2$ and $f(20) = 3$.

Given a positive number n, there always exists at least one number k such that $f(k) = n$. We want to know the smallest such k.

Input

The input consists of at most 1000 test cases, each on a separate line. Each test case is a positive integer $n < 2^{63}$.

Output

For each test case, display its number n and the smallest number $k > 1$ such that $f(k) = n$. The numbers in the input are chosen such that $k < 2^{63}$.

Sample Input

```
1
2
3
105
```

Sample Output

```
1 2
2 6
3 12
105 720
```

Problem C
Cellphone Typing

A research team is developing a new technology to save time when typing text messages in mobile devices. They are working on a new model that has a complete keyboard, so users can type any single letter by pressing the corresponding key. In this way, a user needs P keystrokes to type a word of length P.

However, this is not fast enough. The team is going to put together a dictionary of the common words that a user may type. The goal is to reduce the average number of keystrokes needed to type words that are in the dictionary. During the typing of a word, whenever the following letter is uniquely determined, the cellphone system will input it automatically, without the need for a keystroke. To be more precise, the behavior of the cellphone system will be determined by the following rules:

1. The system never guesses the first letter of a word, so the first letter always has to be input manually by pressing the corresponding key.
2. If a non-empty succession of letters $c_1c_2 \ldots c_n$ has been input, and there is a letter c such that every word in the dictionary which starts with $c_1c_2 \ldots c_n$ also starts with $c_1c_2 \ldots c_nc$, then the system inputs c automatically, without the need of a keystroke. Otherwise, the system waits for the user.

For instance, if the dictionary is composed of the words "`hello`", "`hell`", "`heaven`" and "`goodbye`", and the user presses "`h`", the system will input "`e`" automatically, because every word which starts with "`h`" also starts with "`he`". However, since there are words that start with "`hel`" and with "`hea`", the system now needs to wait for the user. If the user then presses "`l`", obtaining the partial word "`hel`", the system will input a second "`l`" automatically. When it has "`hell`" as input, the system cannot guess, because it is possible that the word is over, or it is also possible that the user may want to press "`o`" to get "`hello`". In this fashion, to type the word "`hello`" the user needs three keystrokes, "`hell`" requires two, and "`heaven`" also requires two, because when the current input is "`hea`" the system can automatically input the remainder of the word by repeatedly applying the second rule. Similarly, the word "`goodbye`" needs just one keystroke, because after pressing the initial "`g`" the system will automatically fill in the entire word. In this example, the average number of keystrokes needed to type a word in the dictionary is then $(3 + 2 + 2 + 1)/4 = 2.00$.

Your task is, given a dictionary, to calculate the average number of keystrokes needed to type a word in the dictionary with the new cellphone system.

Input

Each test case is described using several lines. The first line contains an integer N representing the number of words in the dictionary ($1 \leq N \leq 10^5$). Each of the next N lines contains a non-empty string of at most 80 lowercase letters from the English alphabet, representing a word in the dictionary. Within each test case all words are different, and the sum of the lengths of all words is at most 10^6.

Output

For each test case output a line with a rational number representing the average number of keystrokes needed to type a word in the dictionary. The result must be output as a rational number with exactly two digits after the decimal point, rounded if necessary.

Sample input

```
4
hello
hell
heaven
goodbye
3
hi
he
h
7
structure
structures
ride
riders
stress
solstice
ridiculous
```

Output for the sample input

```
2.00
1.67
2.71
```