Problem A: Sorting fractions

Advanced Algorithms for Programming Contests

Restrictions

Time: 2 seconds
Memory: 512 MB

Problem description

You are to write a function that can sort any given list of fractions. To test it, you should sort the list of all fractions built from given lists of numerators and denominators.

Input

The input consists of

- one line containing integers N and M ($1 \leq N, M \leq 10^3$) – the amounts of numerators and denominators, respectively
- one line containing a list of N integers $a_1, ..., a_N$ ($-10^9 \leq a_i \leq 10^9$, $a_i \neq a_j \ \forall i \neq j$) – the numerators
- one line containing M positive integers $b_1, ..., b_M$ ($b_i \leq 10^9$, $b_i \neq b_j$ $\forall i \neq j$) – the denominators.

Output

Output the fractions that can be created using numerators and denominators from the respective lists in ascending order. Each fraction should be displayed in a separate line in the format num/den (as in the sample output). **Don't cancel** any of the fractions, that's a different function's job! If some of the fractions have the exact same value, sort them (internally) by the size of their numerator (in ascending order).

Sample input and output

Input	Output
2 3 3 2 6 4 7	2/7 2/6 3/7 2/4 3/6 3/4

Problem B: Buckets

Advanced Algorithms for Programming Contests

Restrictions

Time: 2 seconds
Memory: 512 MB

Problem description

The **Taj Mahal**, the casino you work in, has installed a new gambling device. It consists of N buckets, numerated from 1 to N, and M unmarked balls. When the device is activated, the balls fall randomly – one after another – into the buckets. Before that happens, the players may place bets on which balls (referring to the order in which they fall) end up in which of the buckets, while paying a small fee for each bet they place.

Since the casino's mathematician assured the manager that the odds of anyone winning would be astronomically small, the casino has set out an inordinate amount of money to anyone who manages to correctly guess what the content of any of the buckets will be once all balls have fallen – with one exception: a player can't bet on a bucket to stay empty.

Your task is now to write a program that first documents the fall of the balls and then checks for each of the bets whether it was won or not.

Input

The input consists of

- one line containing N ($10^3 \leq N \leq 10^4$) – the number of buckets – M ($1 \leq M \leq 10 \cdot N$) – the number of balls – and B ($1 \leq B \leq 10^3$) – the number of bets
- M lines containing the numbers $n_1, ..., n_M$ ($1 \leq n_i \leq N$), with n_i being the number of the bucket the i-th ball fell into.
- B lines describing the bets, each of them beginning with numbers b and a (in this order!) – the number of the bucket in question and the amount of balls the player bets to be there ($1 \leq b \leq N, 1 \leq a \leq 100$), and then continuing with a numbers $t_1 < ... < t_a$ – the numbers of the balls the player expects to be in the bucket.

Output

For each of the bets, output ”CORRECT” if it was entirely correct and ”INCORRECT” otherwise.

Sample input and output

Input	Output
1000 3 3 373 106 455 753 2 1 3 106 1 2 373 1 3	INCORRECT CORRECT INCORRECT

Problem C: Categories

Advanced Algorithms for Programming Contests

Restrictions

Time: 2 seconds
Memory: 512 MB

Problem description

We want to categorize a given set of words by the letters they begin and end with. For any category, specified by the begin- and the end-character, you are to find and output all words in it.

Input

The input consists of

- one line containing N ($1 \leq N \leq 10^5$) – the total number of words in the text – and M ($1 \leq M \leq 26^2$) – the number of testcases
- one line containing N strings, consisting of at most 20 letters each. You may assume all strings to only contain lowercase letters.
- M lines containing the testcases, each consisting of two lowercase letters, the begin- and end-character of all words in the category in question, respectively. The testcases are guaranteed to be pairwise distinct.

Output

For each of the testcases, output all words of the specified category in lexicographical order, separated by spaces. If a word appears multiple times, output it only once. If the specified category happens to be empty, output **"Empty category!"**.

Sample input and output

Input	Output
3 2 coin clown car a a c n	Empty category! clown coin

Problem D: Frequency List

Advanced Algorithms for Programming Contests

Restrictions

Time: 2 seconds
Memory: 512 MB

Problem description

You are to create a frequency list for a given text, i.e. you are to calculate the frequencies of all words in the text and display each of the words together with its frequency.

Input

The input consists of

- one line containing N ($1 \leq N \leq 10^5$) – the number of words in the text
- one or more lines containing a text consisting of exactly N words each of length not exceeding 20.

Output

Output the occurring words in lexicographical order, each in a separate line and followed by its frequency. While words in the text may contain capital letters, you are to transform all of them (even names) into lowercase (thereby identifying words consisting of the same letters in different capitalization). Also, you will need to cut off commas and periods as well as exclamation and question marks from the ends of some of the words.

Sample input and output

Input	**Output**
3 Chocolate, chocolate, chocolate!	chocolate 3
4 Adam eats many zebras.	adam 1 eats 1 many 1 zebras 1

Problem E: Cookie selection 2

Algorithms for Programming Contests

Restrictions

Time: 2 seconds
Memory: 512 MB

Problem description

As chief programmer at a cookie production plant you have many responsibilities, one of them being that the cookies produced and packaged at the plant adhere to the very demanding quality standards of the Nordic Cookie Packaging Consortium (NCPC).

At any given time, your production line is producing new cookies which are stored in a holding area, awaiting packaging. From time to time there are also requests from the packaging unit to send a cookie from the holding area to be packaged. Naturally, you have programmed the system so that there are never any packaging requests sent if the holding area is empty. What complicates the matter is the fact that representatives of the NCPC might make a surprise inspection to check if your cookies are up to standard. Such an inspection consists of the NCPC representatives demanding that the next few cookies sent to the packaging unit instead be handed over to them; if they are convinced that these cookies look (and taste) the same, you pass the inspection, otherwise you fail.

Since you do not have the time to always be on the lookout for cookie craving NCPC inspectors, you need there to be an automatic mechanism that decides which cookie should be sent to packaging. Fortunately, a while ago the production plant invested in a measurement thingamajig capable of measuring cookie diameters with a precision of 1 nanometer (nm). Back then you started to always send out the cookie of median size, hoping it would please them, but unfortunately this often lead to them complaining about the cookies being too small, so now you finally decided to adjust that mechanism. From now on, instead of the median it should always send the *largest* cookie from the holding area that is *matched or exceeded* in size by *at least a third* of the holding area's current cookie population to the packaging unit on request.

Input

Each line of the input contains either a positive integer d, indicating that a freshly baked cookie with diameter d nm has arrived at the holding area, or the symbol '#', indicating a request from the packaging unit to send a cookie for packaging. There are at most 600 000 lines of input, and you may assume that the holding area is empty when the first cookie in the input arrives to the holding area. Furthermore, you have read somewhere that the cookie oven at the plant cannot produce cookies that have a diameter larger than 30 centimeters (or $3 \cdot 10^8$ nm).

Output

A sequence of lines, each indicating the diameter in nm of a cookie sent for packaging, in the same order as they are sent.

Sample input and output

Input	**Output**
1 2 3 4 5 6 # # # # 10 #	5 4 3 6 10