An important notion in Computer Science is the syntactical concept of terms. Examples of terms can be:

- `a`
- `b`
- `f(a,b)`
- `s(a)`
- `g(f(a,b),b,s(a))`

Sets of terms are usually defined by *induction*. In such a schema, a set of terms is seen in a constructive way: each element of an inductively defined set is either *constructed from* simpler elements of the set or a *basic* element. For instance, `a` is a basic element, `g(f(a,b),b,s(a))` is constructed from `f(a,b)`, `b` and from `s(a)` using the constructor `g`. In the same vein, `f(a,b)` is constructed from `a` and from `b` using the constructor `f`.

More formally, an inductive definition of a set T of terms is composed by a non-empty set B of basic elements and a set K of constructors. Then, we say that T is the smallest set X containing B (i.e. $B \subseteq X$) and the elements that respect the following rule:

let be $f \in K$ with an arity of n, and let be n elements of X (say, $a_1 \dots a_n$) then $f(a_1, \dots, a_n)$ must be in X

Let $h_B : B \to \mathbb{N}$ be a total function that maps every symbol of B to a positive integer, and h_K be a total function over K that maps every n-ary constructor to an n-ary function over positive integers.

In such a setting, we define the notion of *natural interpretation* h as a function from T to $\mathbb{N}$ that maps every term $t \in T$ to a positive integer in the following way:

$$\begin{cases} h(a) & = & h_B(a) \quad \text{if} \quad a \in B \\ h(f(a_1 \dots a_n)) & = & h_K(f)(h(a_1) \dots h(a_n)) \end{cases}$$

We say that an inductive definition T paired with a natural interpretation h is *ambiguous* when there exist two terms $t_1, t_2 \in T$ such that $h(t_1) = h(t_2)$. We also say that (T, h) is *incomplete* when there exists a positive integer n such that there is no term t that verifies $h(t) = n$. Finally we say that (T, h) is *regular* if it is neither incomplete nor ambiguous.

Given an inductive definition of a set T of term and a natural interpretation h, your task consists in qualifying if (T, h) is ambiguous, incomplete, both or regular.

In the context of this problem, we will only consider simple interpretations. As a consequence, elements in h_K are simple functions defined by the following grammar:

$$f ::= (f + f) \mid (f * f) \mid var_{id} \mid \mathbb{N}$$

For a p-ary function, the only valid var_{id} are `x1`, `x2` … `xp`, `x1` for the first argument, `x2` for the second argument, and so on. Consider that every component in the definition of a function is separated from the other by a single space. For instance the *successor* function is described by `( x1 + 1 )`.

In order to simplify the problem, you will not have to consider the whole set of natural numbers. You only will have to consider the set $\{N..M\}$ with $0 \leq N < M \leq 30000$, both provided by the input data. Consider also that each constructor have at least one parameter and at most 5 parameters.

Input

The input will contain several test cases, each of them as described below. Consecutive test cases are separated by a single blank line.

The input consists in the following lines:

- the first line contains N and M, in this order and separated by a single space;
- the second line contains a single integer n $(1 \leq n)$ that represents the number of elements of B;
- the next line contains a single integer m $(0 \leq m)$ that is the number of constructors;
- the following m lines introduce the arity of the m constructors. Thus, each line contains a single integer;
- the next n lines define the function h_B. The first of these lines contains an integer x $(= h_B(a))$ related to the first basic element a (remember, $N \leq x \leq M$). And so on;
- the last m lines define h_K. Each line is then the description of a function that respects the grammar exposed above.

Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

The output is organized following one of these four situations:

Case (T, h) is regular: a single line with the word '`REGULAR`'.

Case (T, h) is incomplete: a single line with the word '`INCOMPLETE`', a single space followed by a integer that is the smallest value that cause the incompleteness of (T, h).

Case (T, h) is ambiguous: a single line with the word '`AMBIGUOUS`', a single space followed by a integer that is the smallest value that turns (T, h) ambiguous.

Case (T, h) is both incomplete and ambiguous: the output is, in this case, two lines long. The first line reports the incompleteness along the lines of the second case. The second line reports the ambiguity of (T, h) in the same way as the third case.

Sample Input

```
0 30000
1
1
1
0
( x1 + 1 )

0 30000
1
1
1
1
( x1 + 1 )

0 30000
1
2
1
2
1
( x1 + 1 )
( x1 + x2 )
```

Sample Output

```
REGULAR

INCOMPLETE 0

INCOMPLETE 0
AMBIGUOUS 2
```