The Problem with the Problem Setter (Easy ver.)

The number of students interested to participate in this year's *Intra-BUET Programming Contest* is huge. Since it is very difficult to accommodate such a large number of students in our labs, we have decided to arrange a *Screening Test*. The test will be paper-based and may include as many as 100 analytical problems from as many as 20 categories. I have been assigned the job of setting problems for this test.

At first, the job seemed to be very easy since I was told that I would be provided with a pool of about 1000 analytical problems already divided into appropriate categories. But after getting the problems I discovered that for many problems the original authors were not sure about the appropriate categories and so they wrote down multiple category-names in the category fields. Since in the *Screening Test* a problem cannot be placed under more than one category and the number of problems to be set under each category is fixed, setting problems for this test is not actually easy.

I know that a program can be written that can do the job automatically. But since I don't like writing programs, I seek your help.

Input

The input file may contain multiple test cases. Each test case begins with a line containing two integers: n_k and n_p ($2 <= n_k <= 20$, $n_k <= n_p <= 1000$) where n_k is the number of categories and n_p is the number of problems in the pool. The second line contains n_k positive integers where the i-th integer specifies the number of problems to be included in category i ($1 <= i <= n_k$) of the test. You may assume that the sum of these n_k integers will never exceed 100. The j-th ($1 <= j <= n_p$) of the next n_p lines contains the category information of the j-th problem in the pool. A category specification for a problem start with a positive integer not greater than n_k, specifying the number of categories in one of which this problem can be included, followed by the category numbers. Category numbers are positive integers not greater than n_k.

A test case containing two zeros for n_k and n_p terminates the input.

Output

For each test case in the input print a line containing either 1 or 0 depending on whether or not problems can be successfully selected form the pool under the given restrictions (1 for success and 0 for failure).

Sample Input

```
3 15
3 3 4
2 1 2
1 3
1 3
1 3
```

```
1 3
3 1 2 3
2 2 3
2 1 3
1 2
1 2
2 1 2
2 1 3
2 1 2
1 1
3 1 2 3
3 15
7 3 4
2 1 2
1 1
1 2
1 2
1 3
3 1 2 3
2 2 3
2 2 3
1 2
1 2
2 2 3
2 2 3
2 1 2
1 1
3 1 2 3
0 0
```

Sample Output

```
1
0
```