Problem B: Cuckoo for Hashing

An integer hash table is a data structure that supports insert, delete and lookup of integer values in constant time. Traditional hash structures consist of an array (the *hash table*) of some size n, and a *hash function* $f(x)$ which is typically $f(x) = x \mod n$. To insert a value x into the table, you compute its *hash value* $f(x)$ which serves as an index into the hash table for the location to store x. For example, if $x = 1234$ and the hash table has size 101, then 1234 would be stored in location $22 = 1234 \mod 101$. Of course, it's possible that some other value is already stored in location 22 ($x = 22$ for example), which leads to a *collision*. Collisions can be handled in a variety of ways which you can discuss with your faculty advisor on the way home from the contest.

Cuckoo hashing is a form of hashing that employs two hash tables T_1 and T_2, each with its own hash function $f_1(x)$ and $f_2(x)$. Insertion of a value x proceeds as follows: you first try to store x in T_1 using $f_1(x)$. If that location is empty, then simply store x there and you're done. Otherwise there is a collision which must be handled. Let y be the value currently in that location. You replace y with x in T_1, and then try to store y in T_2 using $f_2(y)$. Again, if this location is empty, you store y there and you're done. Otherwise, replace the value there (call it z) with y, and now try to store z back in T_1 using $f_1(z)$, and so on. This continues, bouncing back and forth between the two tables until either you find an empty location, or until a certain number of swaps have occurred, at which point you *rehash* both tables (again, something to discuss with your faculty advisor). For the purposes of this problem, this latter occurrence will never happen, i.e., the process should always continue until an empty location is found, which will be guaranteed to happen for each inserted value.

Given the size of the two tables and a series of insertions, your job is to determine what is stored in each of the tables.

(For those interested, cuckoo hashing gets its name from the behavior of the cuckoo bird, which is known to fly to other bird's nests and lay its own eggs in it alongside the eggs already there. When the larger cuckoo chick hatches, it pushes the other chicks out of the nest, thus getting all the food for itself. Gruesome but efficient.)

Input

Input for each test case starts with 3 positive integers n_1 n_2 m, where n_1 and n_2 are the sizes of the tables T_1 and T_2 (with $n_1, n_2 \leq 1000$ and $n_1 \neq n_2$) and m is the number of inserts. Following this will be m integer values which are the values to be inserted into the tables. All of these values will be non-negative. Each table is initially empty, and table T_i uses the hash function $f_i(x) = x \mod n_i$. A line containing 3 zeros will terminate input.

Output

For each test case, output the non-empty locations in T_1 followed by the non-empty locations in T_2. Use one line for each such location and the form `i:v`, where `i` is the index location of the table, and `v` is the value stored there. Output values in each table from lowest index to highest. If either table is empty, output nothing for that table.

Sample Input

```
5 7 4
8 18 29 4
6 7 4
8 18 29 4
1000 999 2
1000
2000
0 0 0
```

Sample Output

```
Case 1:
Table 1
3:8
4:4
Table 2
1:29
4:18
Case 2:
Table 1
0:18
2:8
4:4
5:29
Case 3:
Table 1
0:2000
Table 2
1:1000
```