beecrowd | 1256

Hash Tables

By Neilor Tonin, URI Brasil

Timelimit: 1

Hash tables are used to store elements based on the absolute value of their keys and collision handling techniques. To calculate an address where should be stored a determinated key, it uses a function called hash function which transforms the key in one of the available addresses in the table.

Let's assume that an application uses a hash table with 13 base addresses (indexes 0 through 12) and uses the dispersion function h(x) = x mod 13, where x is the key whose base address would be calculated.

If x = 49, the hash function will return 10, indicating the location (address) where this key should be stored. If we needed insert the key 88 in the same application, the calculation returns the same value 10, a collision will occurs. Treatment of collisions is used to solve conflicts in cases where more than one key is mapped to the same address. This treatment may consider key address recalculation or exterior chaining.

So the teacher asked you to write a program that calculates the address for many keys in some tables, with functions of spreading and treatment of collision by exterior chaining.

Input

The input contains many test cases. The firs line of input contains an integer N indicating the number of test cases. Each test case is composed by two lines. The first one contains a integer **M** (1 ≤ **M** ≤ 100) that indicates the number of base addresses in the table (usually a prime number) followed by an space and a integer **C** (1 ≤ **C** ≤ 200) that indicates the among of keys to be stored. The second one contains each one of the C keys (with value between 1 and 200), separated by an white space.

Output

The output must be printed like the following examples, where the quantity of lines of each test case is determinated by the value of **M**. A blank line should separate each set of output.

Sample Input

```
2
13 9
44 45 49 70 27 73 92 97 95
7 8
35 12 2 17 19 51 88 86
```

Sample Output

```
0 -> \
1 -> 27 -> 92 -> \
2 -> \
3 -> \
4 -> 95 -> \
5 -> 44 -> 70 -> \
6 -> 45 -> 97 -> \
7 -> \
8 -> 73 -> \
9 -> \
10 -> 49 -> \
11 -> \
12 -> \

0 -> 35 -> \
1 -> \
2 -> 2 -> 51 -> 86 -> \
3 -> 17 -> \
```

Sample Input	Sample Output

	`4 -> 88 -> \`
	`5 -> 12 -> 19 -> \`
	`6 -> \`