Prefix lookup is an important part of Router design. Looking at the destination address, a router needs to decide about the potential outbound link. For example, assume a router table consists of the following entries

Prefix	Outbound Link
0*	1,2
11*	1,3
011*	2,3
1110*	4
*	1

Here '*' means following bits are dont care. Now if a destination address (8 bit) takes the form 01101010, then it matches with two prefixes 0*, 011*. Router always takes the longest matching prefix, and thus 01101010 will match with 011*, but not with 0*. On the other hand, address 10101010 does not match with any other prefix except the default prefix, '*'.

Current routers can do this lookup very efficiently, and in this problem, we are not interested about efficient implementation of longest matching prefix.

Given a set of prefix entries, it is significant to know, which prefixes are commonly looked up. In that case, corresponding links can be given special priority (such as more bandwidth share). Here, you need to run a simulation to find the number of times certain prefixes are looked up for a given set of prefixes, and all possible destination addresses. As an example, for 4-bit addresses 0* matches with 6 addresses (0000, 0001, 0010, 0011, 0100, 0101)

Input

Each test case starts with two integers N ($1 \le N \le 20000$), and M ($1 \le M \le 64$). N denotes the number of prefixes whereas M represents the number of bits in an address. Next N lines contains the prefixes (combination of at most M '0', '1's followed by '*'). Along with given prefix entries, you should also assume that there is a default prefix, '*' (matches with all addresses). Next line contains an integer K ($0 < K < 1000$) that represents the number of queries. Following K lines contain the query prefixes (a subset of prefix table entries). Input for each set is followed by a blank line.

Input file terminates with a line containing two zeroes.

Output

For each test case, print the number of matched addresses (different line for each query prefix). Put a blank line after each test case.

Sample Input

```
4 4
0*
11*
011*
1110*
5
*
0*
11*
011*
1110*

4 32
0*
11*
011*
1110*
5
*
0*
11*
011*
1110*

0 0
```

Sample Output

```
4
6
3
2
1

1073741824
1610612736
805306368
536870912
268435456
```