Decode the Strings

Bruce Force has had an interesting idea how to encode strings. The following is the description of how the encoding is done:

Let x_1, x_2 ... x_n be the sequence of characters of the string to be encoded.

1. Choose an integer *m* and *n* pairwise distinct numbers p_1,p_2 ... p_n from the set {*1*, *2* ... *n*} (a permutation of the numbers *1* to *n*).
2. Repeat the following step *m* times.
3. For $1 \le i \le n$ set y_i to x_{p_i}, and then for $1 \le i \le n$ replace x_i by y_i.

For example, when we want to encode the string "hello", and we choose the value *m = 3* and the permutation *2, 3, 1, 5, 4*, the data would be encoded in 3 steps: "hello" → "elhol" → "lhelo" → "helol".

Bruce gives you the encoded strings, and the numbers *m* and p_1 ... p_n used to encode these strings. He claims that because he used huge numbers *m* for encoding, you will need a lot of time to decode the strings. Can you disprove this claim by quickly decoding the strings?

Input

The input contains several test cases. Each test case starts with a line containing two numbers *n* and *m* ($1 \le n \le 80$, $1 \le m \le 10^9$). The following line consists of *n* pairwise different numbers p_1 ... p_n ($1 \le p_i \le n$). The third line of each test case consists of exactly *n* characters, and represent the encoded string. The last test case is followed by a line containing two zeros.

Output

For each test case, print one line with the decoded string.

Example

Input:

```
5 3
2 3 1 5 4
helol
16 804289384
13 10 2 7 8 1 16 12 15 6 5 14 3 4 11 9
scssoet tcaede n
8 12
5 3 4 2 1 8 6 7
encoded?
0 0
```

Output:

```
hello
second test case
encoded?
```