Statement

After correctly locating the network structure, you access a private chat room whose content is encrypted in a very strange way: each message can be decrypted by performing several operations on an initial N-byte key shared by all users.

Each byte of the message is calculated with an operation described by two integers L and R (with L <= R). The result of the operation is equal to (key [L] XOR key [L+1] XOR key [L+2] .... XOR key [R-1] XOR key [R]), where XOR denotes the Exclusive Or operation on integers.

In some cases, the message seems too slow to reconstruct even with your most powerful computers, while the phones of the network members seem to be able to do these same operations in a fraction of a second. Can you effectively implement this algorithm to decipher the messages exchanged from the key and the list of operations to be performed?

Warning: the inputs being of a large size, a naive solution could work on the first (smaller) test sets but not be sufficient for the larger test sets. It will be necessary to redouble ingenuity to decipher the messages!

Data

Input

Line 1: two integers **N** and **M** between 1 and 100,000 separated by a space, respectively the size of the key and the size of the message.

Line 2: integers between 0 and 255 separated by spaces, the bytes of the key

Lines 3 to **M**+2: On the i^{th} line, two integers **L**i and **R**i (with 0 <= **L**i <= **R**i <= **N**-1) corresponding to the operation to be performed to retrieve the i^{th} byte of the message

Output

1 line with 256 integers `N1 ... Ni ... N255` separated by spaces, `Ni` being the number of `i` in the decrypted message.

Example

For the input:

```
5 4

11 22 33 44 55

1 3

0 1

2 2

2 4
```

The output should be:

```
0000000000000000000000000001010001000000000000000000000000
1000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000
00000000000000000000000
```

Indeed:

```
22 xor 33 xor 44 = 27

11 xor 22 = 29

33 = 33

33 xor 44 xor 55 = 58
```

Then, the decrypted message is `27 29 33 58`, hence the expected output.