G. Modulo 3

time limit per test: 4 seconds
memory limit per test: 512 megabytes

Surely, you have seen problems which require you to output the answer modulo $10^9 + 7$, $10^9 + 9$, or 998244353. But have you ever seen a problem where you have to print the answer modulo 3?

You are given a functional graph consisting of n vertices, numbered from 1 to n. It is a directed graph, in which each vertex has exactly one outgoing arc. The graph is given as the array $g_1, g_2, \ldots, g_n$, where g_i means that there is an arc that goes from i to g_i. For some vertices, the outgoing arcs might be self-loops.

Initially, all vertices of the graph are colored in color 1. You can perform the following operation: select a vertex and a color from 1 to k, and then color this vertex and all vertices that are reachable from it. You can perform this operation any number of times (even zero).

You should process q queries. The query is described by three integers x, y and k. For each query, you should:

- assign $g_x := y$;
- then calculate the number of different graph colorings for the given value of k (two colorings are different if there exists at least one vertex that is colored in different colors in these two colorings); since the answer can be very large, print it **modulo** 3.

Note that in every query, the initial coloring of the graph is reset (all vertices initially have color 1 in each query).

Input

The first line contains two integers n and q ($1 \le n, q \le 2 \cdot 10^5$).

The second line contains n integers $g_1, g_2, \ldots, g_n$ ($1 \le g_i \le n$).

The q lines follow. The i-th line contains three integers x_i, y_i and k_i ($1 \le x_i, y_i \le n$; $1 \le k_i \le 10^9$).

Output

For each query, print a single integer — the number of different graph colorings for the given value of k, taken modulo 3.

Examples

input

```
4 5
2 3 1 4
4 3 1
2 1 2
3 4 3
4 1 5
2 4 4
```

output

```
1
2
0
2
1
```

input

```
8 10
7 4 6 8 7 7 1 4
1 7 5
2 3 3
8 6 1
3 1 3
7 2 5
```

```
5 2 4
2 7 4
4 6 5
5 2 3
4 5 1
```

output Copy

```
1
0
1
0
2
1
1
2
0
1
```