BxOI
2019

Setting clocks (clocks)

Idea: KTH Qualifier 2018   Author: Ludo Pulles   Preparation: Marcel Vlastuin

Time limit: 2 s   Memory limit: 512 MB

A friend of you is running an 'Internet of Things' startup which produces clock wearables (formerly known as a "watch"). However, there is a twist to his lucrative business: he has a god mode which enables him to modify the clocks of his customers. All the clocks are given an identification number and only display the hours `00` to `23`, since it has a minimalistic design (less is more!).

When your friend started feeling bored, he challenged you to a test. Repeatedly your friend will add Δ_i hours to all the clocks with an identification number in the range $[A_i, B_i]$ (including A_i, B_i) and you have to tell each time, how many clocks show the time `00`. This wraps back in a 24-hour cycle, so for example adding 5 hours to `22` results in `03`.

Input

The first line contains N: the number of clocks and M: the number of modifications.

The second line contains the initial times of clocks: $h_1, h_2, \ldots, h_N$ separated by one space.

The remaining M lines each contain three numbers, the i^{th} line has Δ_i, A_i and B_i. The integer Δ_i represents the number of hours to add to all the clocks with identification number x for which $A_i \leq x \leq B_i$.

Output

For each of the M modifications, you should output on the i^{th} line the number of clocks that are showing the time `00` when the first i modifications have been performed.

General limits

- $1 \leq N, M \leq 10^5$;
- $h_j \in \{0, 1, 2, \ldots, 23\}$ for all $1 \leq j \leq N$;
- $\Delta_i \in \{1, 2, \ldots, 23\}$ and $1 \leq A_i \leq B_i \leq N$ for all $1 \leq i \leq M$.

Additional constraints

Subtask	Points	Constraints
A	10	$A_i = 1$ and $B_i = N$
B	20	$1 \leq N, M \leq 10^4$
C	30	$\Delta_i = 12$ for $1 \leq i \leq M$; $h_j \in \{0, 12\}$ for $j = 1, \ldots, N$
D	40	No additional constraint

Example 1

Valid for subtasks: A,B,D

`sample1.in`

```
5 4
21 7 2 21 23
3 1 5
7 1 5
12 1 5
11 1 5
```

`sample1.out`

```
2
0
1
0
```

The first modification gives the state

`0 10 5 0 2`,

the second gives

`7 17 12 7 9`,

the third gives

`19 5 0 19 21`,

and the last one modifies the state to

`6 16 11 6 8`.

Example 2

Valid for subtasks: B,D

sample2.in

```
7 3
21 22 23 22 0 1 23
1 2 5
1 1 2
23 2 7
```

sample2.out

```
1
2
2
```

The first modification gives the state

`21 23 0 23 1 1 23`,

the second gives

`22 0 0 23 1 1 23`,

and the last one modifies the state to

`22 23 23 22 0 0 22`.

Example 3

Valid for subtasks: B,C,D

sample3.in

```
6 3
0 12 0 12 12 0
12 1 3
12 2 6
12 3 4
```

sample3.out

```
2
3
1
```

The first modification gives the state

`12 0 12 12 12 0`,

the second gives

`12 12 0 0 0 12`,

and the last one modifies the state to

`12 12 12 12 0 12`.