13123 Christmas Lights

Bill has a Christmas Lights String to decorate his house, made with K lights $L[1], L[2], \ldots, L[K]$ attached in sequence to a wire. The behavior of each light is determined by a programmable controller connected to the wire, turning *on* and *off* lights at every second.

Bill has programmed the controller to change the state of the lights during M seconds. He defines a pair of numbers a_i, b_i with $a_i \leq b_i$, for each second i ($1 \leq i \leq M$). At second 0, the string of lights is initialized with a random *initial configuration* (some lights *on* and other lights *off*). At each second i, from 1 to M, the state of all lights in $L[a_i \mathinner{..} b_i]$ is simultaneously switched from *on* to *off* and vice versa. However, Bill added a curious little feature to the controller's algorithm: whenever the ends $L[a_i]$ or $L[b_i]$ are *off*, just before the above-described switching takes place at time i, some more lights in the string can switch states at moment i. In particular, if $L[a_i]$ is *off* and there is a light, say at l_i, to the left of a_i that is *on* (and all the lights between l_i and a_i are *off*), then the lights in the interval $L[l_i \mathinner{..} a_i - 1]$ will also switch states at moment i. Similarly, if $L[b_i]$ is *off* and there is a light, say at r_i, to the right of b_i that is *on* (and all the lights between b_i and r_i are *off*), then the lights in the interval $L[b_i + 1 \mathinner{..} r_i]$ will also switch states at moment i.

Suppose that a light turned *on* is represented with '`1`' and a light turned *off* is represented with '`0`'. For example, consider $K = 18$, $M = 5$, $a_1 = 5$, $b_1 = 12$, $a_2 = 10$, $b_2 = 11$, $a_3 = 5$, $b_3 = 8$, $a_4 = 3$, $b_4 = 6$, $a_5 = 1$, and $b_5 = 17$, with initial configuration `000110010011100000`. Note that the state of all lights at each second is:

- `000110010011100000` at second 0.
- `000101101100100000` at second 1.
- `000101101011000000` at second 2.
- `000010010011000000` at second 3.
- `001101100011000000` at second 4.
- `110010011100111110` at second 5.

After several days of operation, Bill suspects that he has created a truly awesome algorithm. For this purpose, he would like to run multiple trials, with different initial configurations and parameters a, b, but he is afraid the lights will break due to heavy abuse. Can you help him in building an algorithm for finding the final state of all lights at second M after each trial?

Input

The first line of the input contains a positive integer T indicating the number of test cases. The first line of a test case contains two blank-separated integers K and M ($2 \leq K \leq 10^6$, $0 \leq M \leq 10^4$) indicating, respectively, the number of lights in the string and the number of seconds to consider. The second line contains a hexadecimal string (using digits '`0123456789ABCDEF`') without leading zeros, describing the initial configuration of lights if it is written in binary notation. If the given hexadecimal string requires less than K bits in binary notation, then complete it with leading zeros to reach K digits. Finally follow M lines: line i contains exactly two blank-separated integers a_i and b_i describing the parameters controlling the behavior of the lights at second i ($1 \leq i \leq M$, $1 \leq a_i \leq b_i \leq K$).

Output

For each test case, print a single line with a hexadecimal string (using digits '0123456789ABCDEF') without leading zeros, describing the state of all lights at second M. You must use the same notation used to codify the initial configuration of lights.

Sample Input

```
3
18 5
64E0
5 12
10 11
5 8
3 6
1 17
18 0
0
18 1
0
13 16
```

Sample Output

```
3273E
0
3C
```