EEPROM

The problem is solved using dynamic programming. The data will be written in row-major order. At every instant, the only relevant data (parameters for the subproblem):

- r, c – the row and columns of the cell where the next bit will be written
- hor – whether a horizontal operation was used to set cell (r, s–1)
- mask – bitmask specifying, for each column, whether the lowest already processed cell in the column was changed with a vertical operation

The above parameters give a recursive relation that optimally solves the problem. We can reduce the memory usage by only keeping the last two rows of the table.

TORNJEVI

For each tower we can separately whether it should fire left or right, and up or down. This can be modeled with two Boolean variables per tower. For tower i:

- If the tower fires left, then variable L_i is true
- If the tower fires up, then variable U_i is true
- If the tower fires right, then variable L_i is false, and the expression $\neg L_i$ ("not L_i") is true
- If the tower fires down, then variable U_i is false

The values of some of the variables are predetermined: for example, if two towers are in the same row with no castles between them, then the left tower cannot fire right, and the right tower cannot fire left.

The positions of attackers introduce additional constraints on the values of the variable – every attacker has to be hit by at least one tower. But there are at most two candidate towers for taking out one attacker. It there were three or more, then two of them would be in the same row or column, and one of them would get destroyed. So each attacker either:

- Uniquely determines the value of some variable (when there is only one candidate tower), or
- Says "I have to be hit by one of these towers", setting a logical condition of the form (for example) $L_A \vee \neg U_B$ ("I have to be hit by tower A firing left or tower B firing down")

The problem is now to choose the values of variables L_i and U_i so that all conditions are satisfied. This problem is appropriately named satisfiability. No algorithm is known which solves this problem in the general case, but when each condition contains at most two variables, as in this problem, then it is solvable (this variant is called 2-SAT). In fact, there are multiple algorithms that solve this problem. One of them is: for each variable (whose value still hasn't been determined), assume it is a lie then (recursively) check if this leads to a contradiction – if yes, then the variable must be true. Additionally, assuming the value of a variable can imply the values of other variables. If there is no contradiction, these become determined.