

Czech ACM Student Chapter

Charles University in Prague
Masaryk University
Pavol Jozef Šafárik University in Košice
Slovak University of Technology

Czech Technical University in Prague

Technical University of Ostrava
University of Žilina
Comenius University



CTU Open Contest 2023

Wall

`wall.(c|cpp|py)`, `Wall.(java|kt)`

Tomorrow Programming School is going to decorate the front wall in the entrance hall with a distinctive pattern based on a visually appealing output of an elementary cellular automaton. The designers are going to study the outputs of several cellular automata and choose the one they like the most.

Your task is to write a program that replicates the output of an automaton.

Now, we describe elementary cellular automata and how they work. An elementary cellular automaton is a system consisting of a row of adjacent cells, and a rewriting rule. Each cell is always in one of two states: 0 or 1. The sequence of states of all cells, in the order of cells in the row, is called a generation.

The automaton operates in cycles. In one cycle, the current generation is taken as input and a new generation is calculated by applying the automaton rule to each cell in the current generation. At the end of a cycle, the current generation is replaced by the new generation. Then, another iteration of the cycle can start. The cycle can be repeated for any number of times.

It is assumed that the leftmost and the rightmost cells are also adjacent to other unused cells, which are always in state 0. This assumption ensures the automaton rule can be applied in the same way to all cells in the row. The unused cells do not appear in any input or output.

The state of a particular cell in the new generation is determined by its own state and the state of its two adjacent cells in the current generation, and by the automaton rule.

Let us consider a triple of cells consisting of a particular cell C, and its left and right adjacent cell. There are $2 \cdot 2 \cdot 2 = 8$ possible states, in which this triplet can be in the current generation: 0 or 1 in the left adjacent cell, 0 or 1 in the cell C itself, 0 or 1 in the right adjacent cell. Explicitly, all possible states of this triple of cells may be written as a sequence S of eight binary numbers: $S = (111, 110, 101, 100, 011, 010, 001, 000)$. The middle digit denotes the state of C, and the first and the third digits denote the states of the left and the right adjacent cell to C, respectively.

The automaton rule assigns one bit, 0 or 1, to each of the eight binary numbers in S. The rule is then coded as an 8-bit vector of the bits assigned to binary numbers in S. The sequence S itself is not coded in the automaton rule.

Application of the rule on cell C consists of identifying the binary number in S which corresponds to the states of C and its two adjacent cells in the current generation. The state of C in the next generation is determined by the bit value assigned by the automaton rule to that particular binary number.

There are $2^8 = 256$ different automatons, distinguished by their rule codes. In the input of this problem, the 8-bit vectors representing automaton rules are coded in decimal.

Input Specification

The first line of the input contains two integers, R and K ($0 \leq R \leq 255$, $1 \leq K \leq 200$), the automaton rule coded in decimal and the number of generations, respectively. The second line of the input defines the first generation of the automaton. The cell states are coded by characters, with `'.'` coding 0 and `'X'` coding 1. The row of cells in the first generation is coded as a single string without spaces. The width of the row is at least 1 cell and it does not exceed 250 cells.

Output Specification

The output contains the following K generations produced by the given automaton. The generations are coded and formatted in the same way as the first generation in the input. Each generation occupies one line, there are no empty lines in the output.

Sample Input 1

```
128 5
XXXXXXXXXXXXX
```

Output for Sample Input 1

```
.XXXXXXXXXXX.
..XXXXXXXXX..
...XXXXXXX...
....XXXXX....
.....XXX.....
```

Sample Input 2

```
30 10
...........X...........
```

Output for Sample Input 2

```
..........XXX..........
.........XX..X.........
........XX.XXXX........
.......XX..X...X.......
......XX.XXXX.XXX......
.....XX..X....X..X.....
....XX.XXXX..XXXXXX....
...XX..X...XXX.....X...
..XX.XXXX.XX..X...XXX..
.XX..X....X.XXXX.XX..X.
```