23 Rule 90

11 points

Introduction

Cellular automata (CA) are mathematical models for systems in which simple components (known as cells) act together following certain rules to produce complex behaviour. They can be used to model biological processes, simulate chemical reactions or study physical phenomena.

The simple one-dimensional CA consists of a single row of cells, where each cell can be in one of two possible states (0 or 1), plus a set of rules for evolving the whole CA state. It can be graphically represented as a sequence of numbers, where each number represents a cell value.

To evolve the current state of the CA, the rules act over the cell neighbourhood, that is, the cells placed at the left and right of a given cell. Rule 90 states that each cell's new value is the *exclusive or* of the two neighbouring (left and right) cell values. So, the next state of this particular CA follows this transition table:

Current pattern	000	001	010	011	100	101	110	111
New state for center cell	0	1	0	1	1	0	1	0

It is known as rule 90 because concatenating the new states for center cell of this table results into the binary number 01011010 that equals to decimal number 90.

To make things funnier an exception has been introduced to this rule. For the left-most and right-most cells, just copy the value of the next or previous cell respectively. That is, given the CA state 00110, the next state will be 01111.

The size of our CA is 64 cells and when representing the state 0s must be replaced by "-" and 1s replaced by "*".

Just one final word about complexity. Compare the output of the examples and notice how example 1 ends up forming Sierpinski fractal while example 2 looks like a random form. The output pattern depends critically on our initial conditions. Regular initial conditions provide regular output, but random initial conditions create somehow chaotic output. Hello Complexity World!

Can you write a program that evolves a given cellular automata in a row of a certain number of steps?

Input

The input will be a pair of lines.

The first line contains the original state of the cellular automata represented by 64 digits.

The second line represents the number of steps the cellular automata will execute reporting its output.

Output

A printed line representing the state of the cellular automata for each of the steps executed.

Example 1

Input

```
0000000000000000000000000000001000000000000000000000000000000000
32
```

Output

```
-------------------------------*-------------------------------
------------------------------*-*------------------------------
-----------------------------*---*-----------------------------
----------------------------*-*-*-*----------------------------
---------------------------*-------*---------------------------
--------------------------*-*-----*-*--------------------------
-------------------------*---*---*---*-------------------------
------------------------*-*-*-*-*-*-*-*------------------------
----------------------*--------------*----------------------
---------------------*-*------------*-*---------------------
--------------------*---*----------*---*--------------------
-------------------*-*-*-*--------*-*-*-*-------------------
------------------*-------*-------*-------*------------------
-----------------*-*-----*-*-----*-*-----*-*-----------------
----------------*---*---*---*---*---*---*---*----------------
---------------*-*-*-*-*-*-*-*-*-*-*-*-*-*-*---------------
--------------*-------------------------------*--------------
-------------*-*-----------------------------*-*-------------
------------*---*---------------------------*---*------------
-----------*-*-*-*-----------------------*-*-*-*-----------
----------*-------*---------------------*-------*----------
---------*-*-----*-*-------------------*-*-----*-*---------
--------*---*---*---*-----------------*---*---*---*--------
-------*-*-*-*-*-*-*-*---------------*-*-*-*-*-*-*-*-------
------*--------------*--------------*--------------*-------
```

```
------*-*------------*-*------------*-*------------*-*-------
-----*---*----------*---*----------*---*----------*---*------
----*-*-*-*--------*-*-*-*--------*-*-*-*--------*-*-*-*-----
---*-------*-------*-------*-------*-------*-------*-------*----
--*-*-----*-*-----*-*-----*-*-----*-*-----*-*-----*-*-----*-*---
-*---*---*---*---*---*---*---*---*---*---*---*---*---*---*---*--
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
```

Example 2

Input

```
10101111100001111111111111101111111111101111111111111101110110101
32
```

Output

```
*-*-*****---*************-**********-************-***-**-*-*
----*---**--**-----------*-*--------*-*----------*-*-*-**----
---*-*-*******----------*---*-------*---*---------*------***---
--*----*------**--------*-*-*-*-----*-*-*-*-------*-*----**-**--
-*-*--*-*----****------*-------*---*-------*-----*---*--***-***-
*---**---*--**--**----*-*-----*-*-*-*-----*-*---*-*-*-***-*-*-**
-*-****-*-*********--*---*---*-------*---*---*-*------*-*-----**
*--*--*---*-------***-*-*-*-*-*-----*-*-*-*-*---*----*---*---***
-**-**-*-*-*-----**-*----------*---*--------*-*-*--*-*-*-*-**-*
***-**------*---***--*--------*-*-*-*-------*-----**--------**--
*-*-***----*-*-**-***-*------*-------*-----*-*---****------****-
----*-**--*----**-*-*--*----*-*-----*-*---*---*-**--**----**--**
---*--****-*--***----**-*--*---*---*---*-*-*-*--*******--*******
--*-***--*--***-**--***--**-*-*-*-*-*-*-------***-----****-----*
-*--*-***-***-*-*****-*****-----------*-----**-**---**--**---*-
*-**--*-*-*-*---*---*-*---**----------*-*---***-***-********-*-*
--****-------*-*-*-*---*-****--------*---*-**-*-*-*-*------*----
-**--**-----*-------*-*--*--**------*-*-*--**--------*----*-*---
********---*-*-----*---**-*****----*-----*****------*-*--*---*--
*------**-*---*---*-*-***-*---**--*-*---**---**----*---**-*-*-*-
-*----***--*-*-*-*----*-*--*-******---*-****-****--*-*-***------*
*-*--**-***-------*--*---**--*---**-*--*--*-*--***----*-**----*-
```