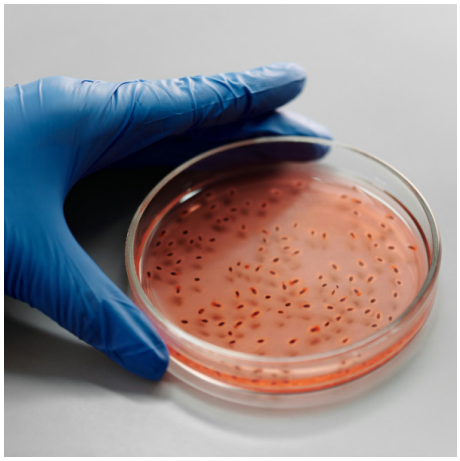
Problem D – Cultura Celular

This is an **output-only problem**.
Unlike the usual problems, where you read data and print output, in this problem you must submit only a single text file.

On a Petri dish represented by an $N \times M$ grid, some cell colonies are initially *alive* and others *dead*. Cells live and reproduce according to the following iterative rule:

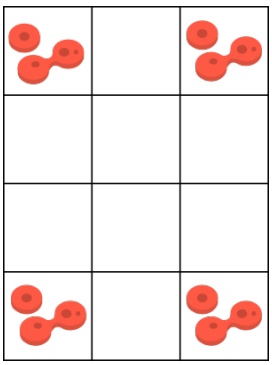


- At every generation (iteration) a dead cell becomes alive if it has at least two *orthogonally adjacent* cells that were alive in the previous generation (due to diffusion of nutrients and biochemical signals). More precisely, for every dead cell we count how many of its four orthogonal neighbors (up, down, left, right) were alive in the previous generation; if at least two of those neighbors are alive, the cell becomes alive in the current generation. An alive cell never dies.
- The process repeats until no more dead cells can be activated, or until all cells on the dish are alive.

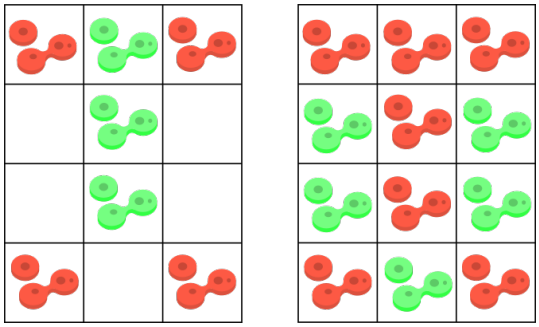
As a biologist you may artificially plant up to K additional cells at the beginning (you may not remove any of the original live colonies). Your goal is to choose those up to K cells so that *after exactly X generations there is still at least one dead cell*, but if the process continues to completion, *every* cell of the grid will eventually become alive.

Example

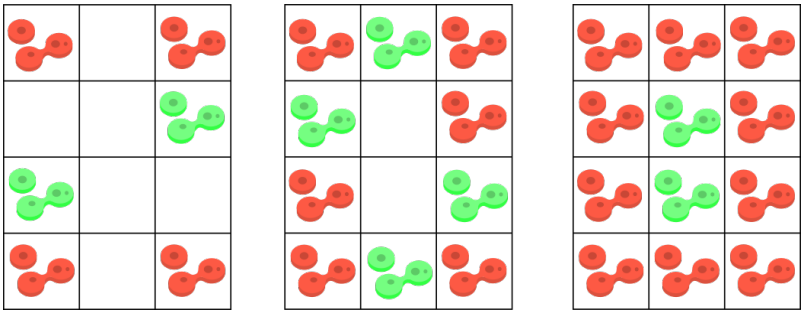
Consider a dish with $N = 3$, $M = 4$ whose initially live cells are:



Suppose $K = 3$ and $X = 2$. If we plant three extra cells in the centre of the grid, the reproduction process lasts exactly two generations, as illustrated below (green cells are those added in the current generation; in the first image they are the three cells we planted):



Because by generation $X = 2$ all cells are alive, this solution is *invalid*. If instead we add the cells shown below, the process takes three generations:



Since at generation $X = 2$ two cells are still dead, and at generation 3 all cells are alive, this solution is *valid*.

Input Format

Each test case starts with four space-separated integers N M K X.

Next follow N lines with M characters each, describing the initial state of the Petri dish:

- `#` denotes an alive cell,
- `.` denotes a dead cell.

For the example above the input would be:

```
3 4 3 2
#.#
...
...
#.#
```

Test Cases

There are 7 test cases, each worth a fixed score (there are no partial scores inside a case). **Treat each test case individually—the cases are not arbitrary, and you should inspect and exploit the structure of each one.**

- **Case 1:** worth 10 points, inp1.txt;
  Description: $N = M = K = 20,\ X = 0$.
- **Case 2:** worth 15 points, inp2.txt;
  Description: $N = 15,\ M = 35,\ K = 25,\ X = 0$.
- **Case 3:** worth 15 points, inp3.txt;
  Description: $N = 30,\ M = 50,\ K = 38,\ X = 0$.
- **Case 4:** worth 15 points, inp4.txt;
  Description: $N = 60,\ M = 80,\ K = 65,\ X = 0$.
- **Case 5:** worth 15 points, inp5.txt;
  Description: $N = M = 16,\ K = 20,\ X = 100$.
- **Case 6:** worth 15 points, inp6.txt;
  Description: $N = 20,\ M = 35,\ K = 40,\ X = 350$.
- **Case 7:** worth 15 points, inp7.txt;
  Description: $N = 50,\ M = 80,\ K = 64,\ X = 2000$.

If you prefer, you can download all cases here: tests.zip.

Output / Submission Format

You must submit a single `.txt` file (the extension matters!). The file must contain *one grid per test case*, each with the exact dimensions of its test case, using `#` for live cells and `.` for dead cells. You may *not* remove original live cells. You may plant at most K additional cells (i.e. change up to K dots into hashes). Your result must satisfy the statement conditions: after exactly X generations there is still at least one dead cell, but if the process is allowed to run to completion, the entire dish eventually becomes alive.

Important notes:

- If you have no solution for a test case, include the original grid.
- If *any* output grid is invalid—wrong size, invalid characters, etc.—the submission is judged `Wrong Answer` and you receive 0 points for *all* test cases.

Sample Validator

To test your submission locally we provide a sample checker that follows the same rules: checker.cpp.

Compile the checker (e.g. `g++ -o checker checker.cpp`) and run it with the input file and the output file produced by your submission: `./checker inpI.txt outI.txt`

Here `inpI.txt` is one of the official inputs, and `outI.txt` is your corresponding output. The checker prints "Correto" if your solution satisfies all conditions, or an error message describing the issue found.

Tips for Generating the Output

Since you must create a `.txt` submission file, the terminal can be handy (this is optional—you can create the file any way you like).

If running your program with command `X` (e.g. for C++: compile with `g++ code.cpp` then run `./a.out`), you can redirect input from a file `inp.txt` and write output to `out.txt` via: `X < inp.txt > out.txt`

It is convenient to generate one `outI.txt` per test case. To concatenate them into a single `out.txt` for submission, you can use:

```
cat out1.txt > out.txt; cat out2.txt >> out.txt; cat out3.txt >> out.txt; cat out4.txt
>> out.txt; cat out5.txt >> out.txt; cat out6.txt >> out.txt; cat out7.txt >> out.txt
```

Organization

APDSI · U.PORTO Faculdade de Ciências Universidade do Porto · Princeton University

High Patronage

Com o Alto Patrocínio de Sua Excelência — O Presidente da República · República Portuguesa, Educação, Ciência e Inovação

Sponsors

Ciência Viva, Agência Nacional para a Cultura Científica e Tecnológica · Fundação Calouste Gulbenkian