M Ű E G Y E T E M 1 7 8 2

3rd BUTE International 24-hour Programming Contest

Qualifying Round Problem Set

http://www.challenge24.org

April 12, 2003

Schönherz Kollégiumi
Számítástechnikai Kör

A verseny fő támogatója a Fornax Rt.

The main sponsor of the contest
is Fornax Co.

Platinum grade sponsors:

Gold grade sponsors:

Silver grade sponsors:

Bronze grade sponsors:

Professional sponsors: BUTE Center of Information Technology, IEEE, John von Neumann Computer Society

General Notes

- There are six problems A, B, C, D, E, and F.
- To solve the problems, you can use any platform, operating system, programming language, development tools, libraries, etc. but you must not use any external human resources.
- When the input file is a text file, then the lines are separated by a single line feed character (character code 10), the last line of the file is also terminated by an LF character. The output of your programs also has to follow this convention. Moreover, there cannot be trailing space characters at the end of the lines.
- There are several test cases for each problem. If not written otherwise, then the input files for problem X are called `X-1.in`, `X-2.in`, . . . , where X is one of A, B, C, D, E, and F. The output files produced by your program should be called `X-1.out`, `X-2.out`, etc.
- In some of the problems, where there are only one correct solution for each test case, we have provided the MD5 checksum of the correct solution. If there is a file `X-1.out.md5` besides the input file `X-1.in`, then this is the MD5 checksum of the correct output for `X-1.in`.
- In Problem B, the correct output could be very large. Therefore you do not have to submit the output files `B-1.out`, `B-2.out`, . . . , but only the MD5 checksums of these files. The files containing the checksum should be called `B-1.out.md5`, `B-2.out.md5`, etc.
- We have provided some sample input/output files for the problems. The first sample input file for Problem X is called `sample-X-1.in`, a correct solution for this test case is given in the file `sample-X-1.out`. In some cases, the MD5 checksum of the output is given in the file `sample-X-1.out.md5`.
- To submit your solution, you should login at `http://www.challenge24.org/reg/login.php` using your team name and password. Then follow the *Electronic contest* link.
- You are required to submit exactly *one .ZIP file* called `X.zip` for each problem X. This .ZIP file should contain the output files for each test case of the given problem. Please note: using .ZIP compression is a must! Multiple uploads per problem is allowed, however, only the last submission is judged.
- If your output is correct for a test case, then you will get the points shown in the table below. You get the points even if your solution for the other test cases are incorrect. Note that we check only your *last* submission for the problem. Therefore if you submit a correct solution, and later you submit an incorrect solution for the test case, then you will get no points for this test case.

Problem A	**Problem B**	**Problem C**	**Problem D**	**Problem E**	**Problem F**
`A-1.in`: 10	`B-1.in`: 10	`C-1.in`: 5	`D-1.in`: 10	`E-1.in`: 10	`F-1.in`: 10
`A-2.in`: 10	`B-2.in`: 10	`C-2.in`: 5	`D-2.in`: 10	`E-2.in`: 10	`F-2.in`: 10
`A-3.in`: 10	`B-3.in`: 25	`C-3.in`: 5	`D-3.in`: 10	`E-3.in`: 10	`F-3.in`: 10
`A-4.in`: 10	`B-4.in`: 25	`C-4.in`: 10	`D-4.in`: 10	`E-4.in`: 10	`F-4.in`: 10
`A-5.in`: 10		`C-5.in`: 15	`D-5.in`: 10	`E-5.in`: 10	`F-5.in`: 10
`A-6.in`: 10		`C-6.in`: 15	`D-6.in`: 10	`E-6.in`: 10	`F-6.in`: 10
`A-7.in`: 10		`C-7.in`: 15	`D-7.in`: 10	`E-7.in`: 10	`F-7.in`: 10
Total: 70	Total: 70	Total: 70	Total: 70	Total: 70	Total: 70

- The 40 teams scoring the highest points will gain the right to participate in the final at Budapest, Hungary May 16-18, 2003. In case of a tie, the sum of the submission times decides the order. *Submission time* of a problem is the time (in minutes) elapsed between the beginning of the contest (10 am) and the last upload for this problem; or 0 if you received 0 point for the given problem. We always use server time.

- If you find a problem statement ambiguous, questions should be sent directly to `challenge24@challenge24.org`. You must not use `contest2003@challenge24.org`! Questions are answered privately.

- In the first hour of the contest, we also send replies immediately to contest2003@challenge24.org. Later a digest of the answers is sent in every 60 minutes.

Problem A: Letters

Introduction

The archaeologists of The Bandulu Republic organized an expedition to excavate the ancient city of Bandu, the forgotten capital of the Bandulu Kingdom. The expedition was successful and tons of invaluable treasures were recovered. The most interesting find was a large collection of stone tablets with some strange-looking text inscribed on them. The experts say that the letters are familiar, but they do not understand the words, which must be in some unknown language, or might be a coded message. To find the meaning of an unknown text, the first thing to do is to count how many times the different symbols appear, this might give us some useful information on the language. However, there are many tablets, and counting letters is a very boring thing to do, therefore the archaeologists ask you to write a program that does this automatically.

Input

Each test case consists of 2 input files: the description of the letters in the text file, and the image of a stone tablet. The file name of the image file is the same as the name of the text file, but it has a `.png` extension instead of an `.in` extension.

The description of the letters is a text file, which defines exactly how each letter is drawn. The first line contains 3 numbers, n, w, and h. The number n is the number of letters described in the file. Each letter is a black and white bitmap w pixels wide and h pixels tall.

The input file contains $h + 1$ lines for each letter. The first line contains only a single character corresponding to the letter. The next h lines correspond to the h rows of the bitmap, and contain w characters each. These characters can be '`.`' to indicate a white pixel, or '`*`' to indicate a black pixel .

The image file is in PNG format, its size can be up to 1024×1024 pixels. The background of the image is white, and the letters in the image are black. Each letter may appear in the normal position, or it may be rotated: it could be rotated 90 degrees clockwise, it could be rotated 90 degrees counter-clockwise, or it could be turned upside down (rotated 180 degrees).

- Every letter appears exactly as it is defined in the text file, there is no "noise" or "error" in the image.
- The letters do not overlap or touch each other. More precisely, if we put each letter in the image into its $w \times h$ bounding rectangle, then these rectangles do not overlap or touch each other.
- There is no ambiguity in recognizing the letters: a rotated letter cannot be the same as any other letter.

Output

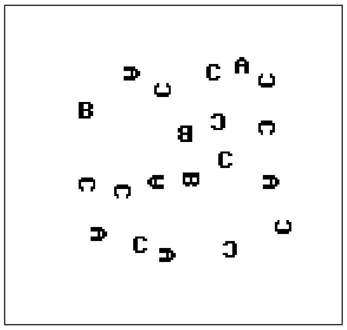
The output is a text file having exactly n lines. Each line begins with a letter, followed by a space and the number of times this letter appears in the image (if the letter does not appear in the image, then this number should be zero). The order of the lines must follow the order in which the letters are defined in the input text file.

Sample Input

Definition of letters (`sample-A-1.in`):

```
3 8 8
A
........
...**...
..****..
.**..**.
.******.
.**..**.
.**..**.
.**..**.
B
........
.*****..
.**..**.
.**..**.
.*****..
.**..**.
.**..**.
.*****..
C
........
..****..
.**..**.
.**.....
.**.....
.**.....
.**..**.
..****..
```

Input image (`sample-A-1.png`):



Sample Output

Number of characters (`sample-A-1.out`):

```
A 6
B 3
C 11
```

Problem B: Expressions

Introduction

In this problem we consider arithmetic expressions that are composed of the following symbols:

- the four binary operators '`+`', '`-`', '`*`', and '`/`',
- the variables '`x`' and '`y`',
- the functions '`sin`' and '`cos`',
- opening and closing parenthesis '`(`' and '`)`'.

Here are some examples for valid expressions:

```
x+x*y+sin(x)   x+(x+x)      (((x)))      sin(x)+cos(y)
 (cos((x)))   ((x)*(x))  (x+y)*(x-y)  sin(sin(sin(x)))
```

Notice that in this problem '`-`' is a binary operator, unary use is not allowed. Therefore '`-x`' and '`x*-y`' are *not* valid expressions. The expression cannot contain spaces. The '`sin`' and '`cos`' functions have to be followed by an argument. The parentheses around the argument of '`sin`' and '`cos`' are obligatory, thus '`sinx`', '`sinx*y`', or '`x+sin`' are *not* valid expressions.

Your job is to write a program that, for a given length ℓ, generates *every* valid expression of length exactly ℓ.

Input

The input file contains only one integer ℓ, the length of the expressions to be generated.

Output

The output of your program is a text file containing every expression of length ℓ exactly once. Each line of this text file should contain one expression, and the lines of the file should be sorted (in ascending ASCII order).

Note: Since the size of the output file can be quite large, in this problem you do not have to submit the output file, but only its MD5 checksum. For example, for test case `B-1.in`, submit the MD5 checksum of the output `B-1.out` in a file called `B-1.out.md5`.

Sample Input

Input file `sample-B-1.in`:

```
3
```

Input file `sample-B-2.in`:

```
6
```

Sample Output

Output file `sample-B-1.out`:

```
(x)
(y)
x*x
x*y
x+x
x+y
x-x
x-y
x/x
x/y
y*x
y*y
y+x
y+y
y-x
y-y
y/x
y/y
```

Output file `sample-B-2.out`:

```
cos(x)
cos(y)
sin(x)
sin(y)
```

Problem C: DNA Sequences

Introduction

"We wish to suggest a structure for the salt of deoxyribose nucleic acid (D.N.A.). This structure has novel features which are of considerable biological interest."
J. D. Watson & F. H. C. Crick, *Nature*, April 25th, 1953

April 25th this year is the 50th anniversary of the seminal article of Watson and Crick in *Nature* that described the structure of DNA, the molecule that contains the genetic information of every living being on Earth. Their discovery was a breakthrough in biology, and it made possible numerous advances in understanding living organisms, including humans. Fifty years later, the Human Genome Project gave us a map of our genes, possibly opening the way for better drugs and treatments.

The success of the Human Genome Project was in large part due to the widespread use of computers and sophisticated software. Mapping the DNA requires the efficient manipulation of massive amounts of data. Computational biology and bioinformatics are new areas of computer science that deal with this kind of problem.

The genetic information in a DNA molecule is coded as a sequence of *bases*. There are four different bases: adenine (A), cytozine (C), guanine (G), and thymine (T). Determining the sequence of bases in a given piece of DNA is called *sequencing* the DNA. Your job is to write a program that helps us in this task.

We have a long piece of DNA that we have to sequence. We have made several measurements, each measurement describes the sequence of bases for some small segment of the DNA. Each such segment has the same length ℓ. The first segment contains the first ℓ bases of the sequence. The second segment starts at the $(\ell - 4)$th base, it overlaps with the first segment on 5 bases. The third segment overlaps with the second segment on 5 bases, and so on. For example, if we have this DNA sequence of 49 bases:

```
ATTCGTACCGGAGTCCCAGACCTCGGGTTAAACACATATAGATGCAGAT
```

and $\ell = 16$, then we have the following 4 measurements:

```
ATTCGTACCGGAGTCC          TCGGGTTAAACACATA
           AGTCCCAGACCTCGGG          ACATATAGATGCAGAT
```

Unfortunately, due to some software errors, the order of the segments became mixed up. So we have all the segments, but we do not know which segment corresponds to which measurement. Given the segments obtained by the measurements (in some random order), your program has to recover the original sequence.

Input

The input is a text file. The first line contains two integers n and ℓ, where n is the number of segments in this test case and ℓ is the length of the segments. The next n lines describe the n segments obtained by the measurements (in random order). Each line contains a string of length ℓ, each letter of the string is one of 'A', 'C', 'G', or 'T'.

Output

You have to output the original sequence in a file whose length equals the length of the original sequence. (Notice that a simple calculation shows that the length of the original sequence is $n(\ell - 5) + 5$). The output file should contain only the 'A', 'C', 'G', or 'T' characters. Do *not* terminate the file with a new line character. If there are multiple possible solutions (the original sequence cannot be unambiguously recovered), then you can output any correct solution.

Sample Input

`sample-C-1.in`:

```
7 12
CACAGTGAGGCT
AGGCTTCAAGCA
TTAGAACCATCC
GGAGGCCCACAG
CATCCTTAGGCT
AGGCTATGGAGG
AGGCTTATTAGA
```

Sample Output

There are two correct possibilities for `sample-C-1.out`:

```
AGGCTATGGAGGCCCACAGTGAGGCTTATTAGAACCATCCTTAGGCTTCAAGCA
```

and

```
AGGCTTATTAGAACCATCCTTAGGCTATGGAGGCCCACAGTGAGGCTTCAAGCA
```

Problem D: Island

Introduction

Although Tikubulu Island is quite small, it was the scene of countless bloody wars throughout the centuries. There has been peace for some years, but the island is divided into several small countries. Now is a good time to draw a map of the island, since we hope that the borders of the countries will stay the same for some time. When drawing the map, we should follow these rules:

- The color of the sea is blue (RGB: 0,0,255).
- The border between two countries, or between a country and the sea is marked by a black line (RGB: 0,0,0).
- The color of a country with area less than 500 km^2 is orange (RGB: 255,128,0).
- The color of a country with area between 500 km^2 and 1000 is red (RGB: 255,0,0).
- The color of a country with area greater than 20000 km^2 is yellow (RGB: 255,255,0).

If the area of the country is between 1000 km^2 and 20000 km^2, then its color is determined by the number of vertices on its boundary:

- A country the boundary of which has 3 vertices (i.e., it is a triangle) should be colored green (RGB: 0,255,0).
- A country the boundary of which has 4 vertices should be colored cyan (RGB: 0,255,255).
- A country the boundary of which has 5 vertices should be colored brown (RGB: 128,0,0).
- A country the boundary of which has 6 vertices should be colored gray (RGB: 192,192,192).
- A country the boundary of which has 7 or more vertices should be colored magenta (RGB: 255,0,255).

Input

The input is a text file containing the description of the map of an island. The first line contains an integer n, which is the number of vertices in the map. The next n lines contain two integers each, the x- and y-coordinates of the vertices (in km). These n lines are followed by a line containing an integer m, the number of boundary line segments in the map. Each boundary line segment connects two vertices. The boundary line segments do not cross each other, and each segment separates two countries, or a country and the sea. The last m lines of the input file describe the m boundary line segments of the map. Each line contains two integers, the end vertices of the segment (the vertices are numbered 0, 1, …, $n-1$, the coordinates of vertex 0 appears first in the input file).

It can be assumed that there is no country which is completely surrounded by some other country. During the history of Tikubulu Island, if some country A completely surrounded some other country B, then sooner or later country A eliminated country B.

Output

You have to output the map of the island in a 1024×1024 PNG file. The color depth, palette, etc. of the file can be arbitrary, as long as it correctly represents the colors of the map. The recommended format is true color. The file name of the output for test case `D-1.in` should be `D-1.png`. Each pixel is 1 km, the coordinates of the pixel in the upper left corner are $(0, 0)$. The x-axis is horizontal, the y-axis is vertical.

There is more than one way of drawing a line between two points, therefore there is more than one correct solution. When your solution is judged, such errors will be tolerated.

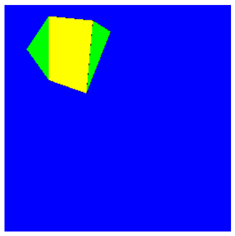
Sample Input

`sample-D-1.in`:

```
6
100 200
200 50
200 340
400 70
370 400
480 120
8
0 1
1 2
0 2
1 3
2 4
3 4
3 5
4 5
```

Sample Output

`sample-D-1.png`:



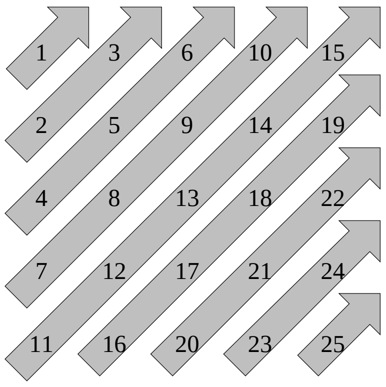
Problem E: Crypto

Introduction

The Bandulu Secret Service has just intercepted a secret message of the Tuluvu army. It is extremely important to decode the message, it may determine the outcome of the Bandulu–Tuluvu war. Fortunately, we know how the Tuluvu army encrypts its messages, your job is to write a program that decrypts the secret message.

The encryption process consists of the following steps.

Step 1. We assume that the length ℓ of the original text is a perfect square, that is, $\ell = n \times n$ for some integer n. Write the message into an n times n matrix: fill the first row, starting from the left going to the right, then fill the second row, and so on. Read the characters "diagonally", in the order shown on the figure: first read the upper left character, then read the characters under the diagonal arrows shown in the figure.



Step 2. In this step we reorder the character sequence obtained in Step 1. We repeat the following three steps ℓ times:

(a) Write down the first character of the sequence, and delete this character from the sequence.

(b) If the length of the remaining sequence is longer than 12 characters, then remove the first 12 characters of the sequence, and append it to the end of the sequence.

(c) If the length of the sequence is longer than 1, then swap the first two characters.

Step 3. The character sequence created in Step 2 can be interpreted as a sequence of integers a_1, a_2, $\ldots$, a_ℓ, where each a_i is between 0 and 255. In this step we calculate a sequence b_1, b_2, $\ldots$, b_ℓ using the following formula:

$$\begin{aligned} b_1 &= a_1 \\ b_i &= (a_i + b_{i-1}) \bmod 256 \text{ for } i > 1 \end{aligned}$$

'mod 256' means taking the remainder of the number modulo 256.

Step 4. We calculate a sequence c_1, c_2, $\ldots$, c_ℓ:

$$c_i = \begin{cases} (b_i + i) \bmod 256 \text{ if } i \text{ is a prime number} \\ b_i \text{ otherwise.} \end{cases}$$

Recall that a number $p > 1$ is a prime number if and only if its only divisors are p and 1. Note that 1 is *not* a prime number.

Step 5. Each number c_i is transformed into two characters. The first character is determined by the upper 4 bits (MSB) of the number, the second character is determined by the lower 4 bits (LSB). The upper 4 bits represent a number between 0 and 15. If this number is 0, then first character is ’`A`’; if this number is 1, then the character is ’`B`’; …; if this number is 15, then the character is ’`P`’. The second character is determined in a similar fashion, based on the value represented by the lower 4 bits of c_i.

Example

For example, consider the message ’`CHALLENGE24-2003`’. In Step 1 we get the matrix

```
C  H  A  L
L  E  N  G
E  2  4  -
2  0  0  3
```

therefore we obtain the sequence ’`CLHEEA22NL04G0-3`’. We begin Step 2 with this sequence. Repeating steps (a)-(c), we get the following sequences:

```
CLHEEA22NL04G0-3
-03LHEEA22NL04G
G403LHEEA22NL0
4003LHEEA22NL
003LHEEA22NL
30LHEEA22NL
L0HEEA22NL
H0EEA22NL
E0EA22NL
E0A22NL
A022NL
202NL
20NL
N0L
L0
0
```

For example, starting with ’`CLHEEA22NL04G0-3`’, the first character ’C’ is deleted, then the first 12 characters ’`LHEEA22NL04G`’ are removed from the beginning, and appended to the end, gives ’`0-3LHEEA22NL04G`’. Finally, in step (c), the first two characters are swapped, which is how we get the sequence ’`-03LHEEA22NL04G`’. Since in step (a) we write down the first character of the current sequence, getting ’`C-G403LHEEA22NL0`’. This is equivalent to a sequence of numbers a_1, a_2, …, a_{16}:

$$67, 45, 71, 52, 48, 51, 76, 72, 69, 69, 65, 50, 50, 78, 76, 48.$$

Therefore the sequence b_1, b_2, …, b_{16} constructed in Step 3 is:

$$67, 112, 183, 235, 27, 78, 154, 226, 39, 108, 173, 223, 17, 95, 171, 219.$$

In Step 4 we modify the value only of the 2th, 3rd, 5th, 7th, 11th, 13th numbers (these are the prime numbers between 1 and 16). Therefore the sequence c_1, c_2, …, c_{16} is

$$67, 114, 186, 235, 32, 78, 161, 226, 39, 108, 184, 223, 30, 95, 171, 219,$$

or in hexadecimal form:

```
0x43, 0x72, 0xba, 0xeb, 0x20, 0x4e, 0xa1, 0xe2, 0x27, 0x6c, 0xb8, 0xdf, 0x1e, 0x5f, 0xab, 0xdb.
```

The upper four bits of `0x43` represents 4, the lower four bits of `0x43` represents 3, therefore Step 5 transforms `0x43` to ’`ED`’. Similarly transforming the other numbers gives the sequence

```
EDHCLKOLCAEOKBOCCHGMLINPBOFPKLNL
```

which will be the encoded form of the original message ’`CHALLENGE24-2003`’.

Sample Input

`sample-E-1.in`:

```
EDHCLKOLCAEOKBOCCHGMLINPBOFPKLNL
```

Sample Output

`sample-E-1.out`:

```
CHALLENGE24-2003
```

Problem F: Ecology

Introduction

The Great Plain of Yukuruku is not a very interesting place. There is grass, there are rabbits, and there are foxes, but there is nothing else. However (for some reason not detailed here), it is very important to keep track of the rabbits and the foxes. Unfortunately, the number of rabbits and foxes is always changing, since the foxes eat the rabbits, and the rabbits breed like, well, like rabbits. You have to write a program that simulates the ecosystem of the plains.

The Great Plain of Yukuruku is modeled by a $n \times m$ matrix, where each cell is in one of three states: either there is grass, or there is a rabbit, or there is a fox. In the input you are given the initial state of each cell. The cells change according to the following three rules:

- If a cell contains grass, and it has a neighbor containing a rabbit, then the rabbit eats the grass, and this cell will also contain a rabbit.
- If a cell contains a rabbit, and it has a neighbor containing a fox, then the fox eats the rabbit, and this cell will also contain a fox.
- If a cell contains a fox, and it does not have a neighbor containing a rabbit, then the fox dies of hunger, and the cell will contain grass.

The neighbors of a cell also include those cells that share only a corner with the cell. Therefore a cell can have at most 8 neighbors (if the cell is on the boundary of the matrix, then it has less than 8 neighbors).

Given the initial state of the matrix, the simulation goes as follows. First we assign the value $f(i, j)$ to the cell in the ith row and jth column, where $(1 \leq i \leq n,\ 1 \leq j \leq m)$

$$f(i, j) = 23i + 87j + 19i^2 + 61j^2 + 13i^3 + 31j^3 \bmod 131,$$

which is a number between 0 and 130 (mod131 means the remainder of the number modulo 131). In the first step we apply the three rules to those cells whose $f(i, j)$ value is 0, in the second step we apply the rules to those cells with value 1, ..., in the 131th step we apply the rule to the cells with value 130. Then we start again: in the 132th step we apply the rule to the cells with value 0, and so on.

In each step, the rules are applied to the selected cells in parallel. This means that for each cell, first we have to check its neighbors and decide what will be the state of the cell in the next step. When we know the next state of each cell, then the cells change their value at once. Therefore it is possible, for example, that a cell with grass and a cell with a fox are neighbors, and in the same step the grass is eaten by a rabbit, and the fox dies of hunger.

Input

The input is a text file describing the initial state of the cells. The first line contains three integers (separated by one space): the number of rows n, the number of columns m, and the length of the simulation ℓ. The next n lines contain m character each, describing the n rows of the initial matrix. The characters in the row can be only '`.`' (grass), '`!`' (rabbit), or '`*`' (fox).

Output

For each test case, you have to output a text file, and 6 images in PNG format. The first ℓ lines of the text file show how the number of grass, rabbits, and foxes change during the simulation. The ith line $(1 \leq i \leq n)$ contains three numbers separated by one space: the number of cells with grass, with rabbit, and with fox *after* the ith step of the simulation. The last 6 lines of the text file gives some statistics. For each type of cells, it shows at which step the matrix contained the maximum and minimum number of cells from this type. These lines must be in the following format:

```
Minimum number of grass: $x_1$ after step $t_1$.
Maximum number of grass: $x_2$ after step $t_2$.
Minimum number of rabbits: $x_3$ after step $t_3$.
Maximum number of rabbits: $x_4$ after step $t_4$.
Minimum number of foxes: $x_5$ after step $t_5$.
Maximum number of foxes: $x_6$ after step $t_6$.
```

If the maximum/minimum of a type is attained multiple times, then you should output the *first* step when this type reaches its maximum/minimum. Moreover, it is possible that, for example, the minimum number of grass is reached before the first step of the simulation (the number of grass cells never go below the number of grass cells in the initial matrix). In this case, t_1 is zero.

The six images that you have to output describe the state of the matrix after the t_1th, t_2th, ..., t_6th step. For test case `F-1.in`, the output PNG files should be called `F-1-1.png`, `F-1-2.png`, ..., `F-1-6.png`. The width of each image is m pixels, the height is n pixels (it has the same size as the matrix). The pixel in the upper left corner corresponds to first cell of the first row. The color of the pixel is determined by the state of the cell:

- Grass: green pixel (RGB: 0,255,0)
- Rabbit: gray pixel (RGB: 192,192,192)
- Fox: red pixel (RGB: 255,0,0)

The color depth, palette, etc. of the file can be arbitrary, as long as it correctly represents the matrix. The recommended format is true color.

Sample Input

Input file (`sample-F-1.in`):

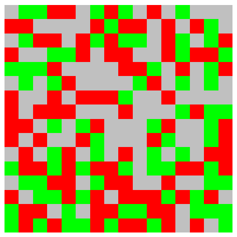
```
16 16 7
!..**!.*.!*!.!!!
**!!!!****!*!*.!
!.**!*.*..!*.!.*
*!!..*.**!!*.**.
...*!!!!**.!*!.*
!.!.*!!!!.*!.!.!
*!!*!***.!!*!!!!
*!**!!!!*!!!.*..
!*!.!.*!!!.*!!.*
*!*!.*.!!!*.!!.*
!*!.*!.!*!.!.!**
.**.*.**.!..**.*
!..**!.*!!!*.!..
*!..*.*!***.*.*!
.**!.!**..**!...
.*.*..*.****!*!.
```

Sample Output

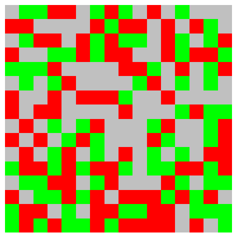
Output text file (`sample-F-1.out`):

```
75 91 90
74 92 90
74 92 90
74 91 91
75 91 90
74 90 92
73 91 92
Minimum number of grass: 73 after step 7.
Maximum number of grass: 75 after step 1.
Minimum number of rabbits: 90 after step 6.
Maximum number of rabbits: 92 after step 2.
Minimum number of foxes: 90 after step 1.
Maximum number of foxes: 92 after step 6.
```

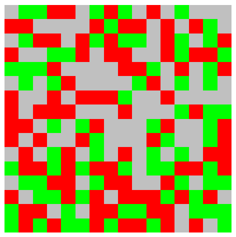
`sample-F-1-1.png`:
Minimum number of grass:
73 after step 7.



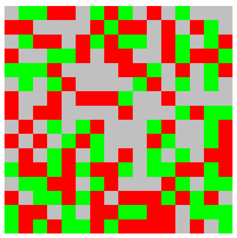
`sample-F-1-2.png`:
Maximum number of grass:
75 after step 1.



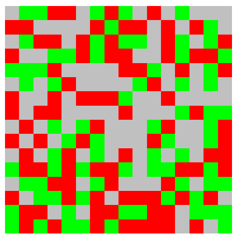
`sample-F-1-3.png`:
Minimum number of rabbits:
90 after step 6.



`sample-F-1-4.png`:
Maximum number of rabbits:
92 after step 2.



`sample-F-1-5.png`:
Minimum number of foxes:
90 after step 1.



`sample-F-1-6.png`:
Maximum number of foxes:
92 after step 6.