Simulating the chip-firing game X31871_en

GRAU-PRO1, FIB (2014-12)

The *chip-firing game* on matrices is a mathematical game that models load balancing issues on a bidimensional structure. Given a square matrix with n rows and n columns, each position (i, j) in it stores an initial amount of chips and a list of the neighboring accessible positions with which the position (i, j) is connected and interacts.

The game considers the spreading of the stored chips over time along the positions of the matrix. The distribution is done according to a local rule to be applied at each position: On every time step, each position having at least as many chips as neighboring accessible positions, sends a chip to each of those neighbors.

Definition 1: In order to simplify the game, we will consider that only the *inner positions* of the matrix play the game. Given a square matrix of size n, the inner positions are those positions (i, j) such that $0 < i < n - 1$ i $0 < j < n - 1$ (i.e., positions neither on the first nor on the last row, neither on the first nor on the last column).

Definition 2: Given a position (i, j) of a matrix, its neighbors are those positions located at distance 1 from it, either horizontally, vertically or in diagonal. Notice that every position (i, j) has, at most, 8 neighboring positions. We will name *accessible neighboring positions*, the subset of neighboring positions which the position can interact with. Every neighboring position can be represented with two characters according to the following schema:

NW	NN	NE
WW		EE
SW	SS	SE

For example, the string "NNSS" (north-north, south-south) would state that the only two accessible neighboring positions are those immediately above and immediately underneath.

To do:
We want to do a sequential simulation of the chip-firing game in which, only one position is updated at every time step. The order in which the positions are updated is defined by a complete column-wise traversal of the matrix. Only the inner positions of the matrix take part on the simulation, being the first one the upmost-leftmost position (i.e., the position $(1, 1)$). From that position on, the simulation proceeds orderly and column-wise over the remaining positions of the matrix, one at a time. A complete simulation of the game in this way is called a *round*.

Given a game matrix, design a program that computes the matrix resulting from the game after playing one round. The program must also indicate whether the resulting matrix is different to the original matrix. Finally, the program must calculate how many chips were moved from inner to outer positions during the played round.

Your program must describe the game matrix using the following type:

```
struct Cell {
    int contents;          // amount of chips
    string neighbors;      // description of the accessible neighboring positions
};

typedef vector< vector<Cell> > Game;
```

Input

The input is composed by a single case, which starts with a natural number $n \geq 3$ defining the size of the squared matrix. Following, n^2 natural numbers represent the amount of chips in every position of the matrix. Then, the accessible neighboring positions of every inner position of the matrix are to be found. Those positions are described by a string according to the description in Definition 2.

Output

Write the number of chips at every position of the matrix after simulating a round and indicate whether some change (w.r.t. the initial game matrix) happened. Write the amount of chips that have been moved from inner to outer positions. Follow the format to be found in the examples below.

Observation

Notice that, even that only the inner positions of the matrix are involved in the simulation, the outer positions can also store chips.

Your code must accomplish with a good programming style. It is up to you to include convenient comments on it. The clarity of the design, the good use of procedures and the style programming will assessed.

Sample input 1

```
4
0 0 0 0
1 1 1 1
0 1 1 0
0 0 0 0
SS SS
SS SS
```

Sample output 1

```
0 0 0 0
1 0 0 1
0 1 1 0
0 1 1 0
changes: yes
new chips outside: 2
```

Sample input 2

```
4
0 0 0 0
1 2 2 1
1 1 1 1
0 0 0 0
SS SS
SS SS
```

Sample output 2

```
0 0 0 0
1 1 1 1
1 1 1 1
0 1 1 0
changes: yes
new chips outside: 2
```

Sample input 3

```
4
0 0 0 0
1 2 1 1
1 1 1 1
0 0 0 0
SSEENN SSSE
SSEE SSEESW
```

Sample output 3

```
0 0 0 0
1 2 1 1
1 1 1 1
0 0 0 0
changes: no
new chips outside: 0
```

Problem information

Author : Maria J. Serna
Translator : Maria Blesa
Generation : 2014-12-16 14:52:52