Algorithmic and Programmation, Project
Autumn 2016
LO27

Matrices and Cellular Automata

Nicolas Gaud and Thomas Martinet

19 octobre 2016

1 Implementation and evaluation of the project

— Group of 1 or 2 persons at most.
— Implementation in C language.
— The project will be documented in a written report that outlines (15 to 20 pages) used data structures, the selected algorithms, optimizations and encountered difficulties. No copy-paste of entire pages of source code. Your report must at least contain :

1. An introduction which recalls the subject, introduces the structure of the report and describes the major points of your work.
2. A table of content
3. The description of each new abstract data types used to implement your library
4. The algorithmic description of each function described below. Each algorithm described in the report have to be documented using the template that have been introduced during the lesson.
5. A conclusion providing an objective evaluation of your personal work, it summarizes your work, the introduced optimizations and the potential **remaining problems**.

— The source codes will be commented and the names of various authors of the project have to be mentioned at the beginning of each source file as well as a textual description specifying its purpose.
— The program will have a minimal GUI console. The GUI is not the core of the project, it should nevertheless be simple and user friendly.
— The report must be in **PDF** format and the corresponding source code have to be delivered no later than **January 2^{nd} 2017 at 6:00pm** by email addressed to `nicolas.gaud@utbm.fr` and `thomas.martinet@utbm.fr`, (Subject : LO27 Project - Group : STUDENTNAME1UPPERCASE and STUDENTNAME2UPPERCASE) in a ZIP or TAR.GZ archive named in the following way :

LO27_STUDENTNAME1UPPERCASE_STUDENTNAME2UPPERCASE.zip

Any delay or failure to comply with these guidelines will be penalized.

A special attention should be paid on the following points when writing the program : the program runs smoothly without bugs (it's better not to provide a feature rather than provide it if it does not work, negative points), readability and clarity of the code (comments and indentation), complexity and efficiency of proposed algorithms, choice of data structures, modularity of the code (development of small functions, distributed in various files comprising functions consistently and appropriately named).

2 Project goal

The goal of this project aims at providing the definition of a new abstract data type called **Matrix** and the set of associated functions to manipulate this new type. In the report, you have to describe all the algorithms of these functions. The objective in terms of C code consist in providing a library of features for handling matrices. You have to provide a complete program enabling a user to test this library in an interactive and practical way.

2.1 Abstract Data Type : Matrix

The matrix data type must at least provide the following functions :

— $newMatrix$: $Boolean[][] \to Matrix$, build a new *Matrix* from its array-based representation ;

— $printMatrix$: $Matrix \to void$, print the content of the specified *Matrix* ;

— $isMatrixEmpty$: $Matrix \to Boolean$, returns true if the specified *Matrix* is empty, false otherwise ;

— $isColumnEmpty$: $Matrix \times Integer \to Boolean$, returns true if the specified *Matrix* column is empty, false otherwise ;

— $isRowEmpty$: $Matrix \times Integer \to Boolean$, returns true if the specified *Matrix* row is empty, false otherwise ;

— $isMatrixSquare$: $Matrix \to Boolean$, returns true if the specified *Matrix* is square, false otherwise ;

— $equalsMatrix$: $Matrix \times Matrix \to Boolean$, true if the two specified *Matrices* are equals ; false otherwise ;

— $sumMatrix$: $Matrix \times Matrix \to Matrix$, computes the boolean addition (OR operator) of the two specified *Matrices* ;

$$sumMatrix(m_1, m_2) \Leftrightarrow m_1 + m_2 \Leftrightarrow m_1 \text{OR } m_2$$

— $mulMatrix$: $Matrix \times Matrix \to Matrix$, computes the boolean product (AND operator) between the two specified *Matrices* ;

$$mulMatrix(m_1, m_2) \Leftrightarrow m_1 \times m_2 \Leftrightarrow m_1 \text{AND } m_2$$

— $andColSequenceOnMatrix : Matrix \rightarrow Matrix$, given in input a matrix with M rows and N columns, this function computes a matrix with M rows and $N-1$ columns where each element corresponds to the AND boolean operation between two successive horizontal elements in the input matrix

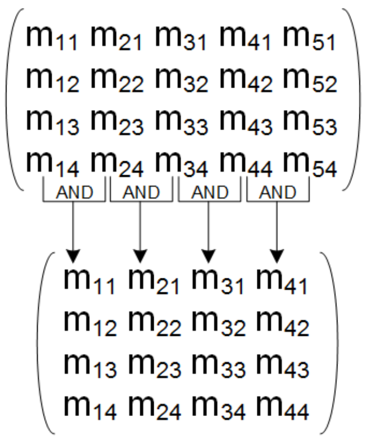


FIGURE 1 – Overall behavior of $andColSequenceOnMatrix$

Example : $M_{output_{11}} = M_{input_{11}}$ AND $M_{input_{21}}$

— $orColSequenceOnMatrix : Matrix \rightarrow Matrix$, given in input a matrix with M rows and N columns, this function computes a matrix with M rows and $N-1$ columns where each element corresponds to the OR boolean operation between two successive horizontal elements in the input matrix

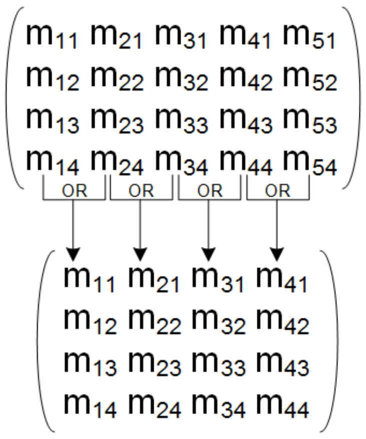


FIGURE 2 – Overall behavior of $orColSequenceOnMatrix$

Example : $M_{output_{11}} = M_{input_{11}}$ OR $M_{input_{21}}$

— *andRowSequenceOnMatrix* : $Matrix \to Matrix$, given in input a matrix with M rows and N columns, this function computes a matrix with $M-1$ rows and N columns where each element corresponds to the AND boolean operation between two successive vertical elements in the input matrix

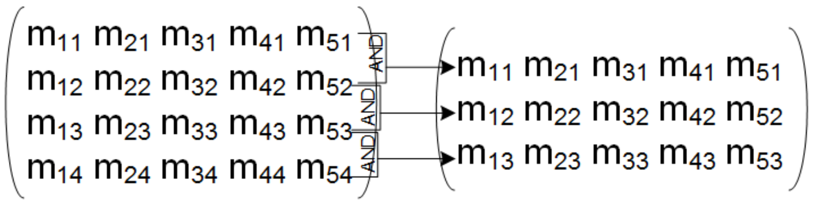


Figure 3 – Overall behavior of *andRowSequenceOnMatrix*

Example : $M_{output_{11}} = M_{input_{11}}$ AND $M_{input_{12}}$

— *orRowSequenceOnMatrix* : $Matrix \to Matrix$, given in input a matrix with M rows and N columns, this function computes a matrix with $M-1$ rows and N columns where each element corresponds to the OR boolean operation between two successive vertical elements in the input matrix

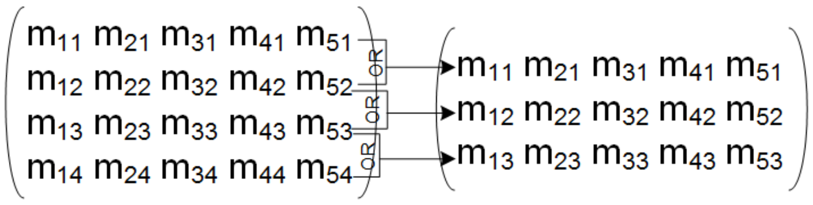


Figure 4 – Overall behavior of *orRowSequenceOnMatrix*

Example : $M_{output_{11}} = M_{input_{11}}$ OR $M_{input_{12}}$

— *applyRules* : $Matrix$(Input)$\times$$Integer$(Rule ID)$\times$$Integer$(number of times) $\to$ $Matrix$, applying the rule identified by the first specified integer (Rule ID) to the specified input matrix (Input) the number of times specified by the second integer (number of times), and it returns the corresponding resulting matrix. More details in the next section.

3 The behavior of the *applyRules* function

A complete description of the desired behavior is provided in the following paper (without considering the sweeper's algorithm) :
Theory and Applications of Two-dimensional, Null-boundary, Nine-Neighborhood, Cellular Automata Linear rules,
see `https://arxiv.org/ftp/arxiv/papers/0804/0804.2346.pdf`

64 $(1000000)_2$	128 $(10000000)_2$	256 $(100000000)_2$
32 $(100000)_2$	1 $(1)_2$	2 $(10)_2$
16 $(10000)_2$	8 $(1000)_2$	4 $(100)_2$

Table 1 – Rules naming convention, integer value and corresponding binary representation

Direction of the translation of the image	**Rule**
Top	8
Bottom	128
Left	2
Right	32
Top-Left	4
Top-Right	16
Bottom-Left	256
Bottom-Right	34

Table 2 – Translation associated to each fundamental rules

Each rule can be realized by a XOR[1] operation on the input matrix. Each rule is characterized by the neighborhood of the considered cell used to compute the XOR operation. A specific rule naming convention is adopted as described in table 1. The central box represents the current cell (i.e. the cell being considered) and all other boxes represent the eight nearest neighbors of that cell. The number within each box represents the rule number characterizing the dependency of the current cell on that particular neighbor only. Rule 1 characterizes dependency of the central cell on itself alone whereas such dependency only on its top neighbor is characterized by rule 128, and so on. These nine rules are called fundamental rules. In case the cell has dependency on two or more neighboring cells, the rule number will be the arithmetic sum of the numbers of the relevant cells.

Figure 5 illustrates the application of fundamental rules 2 and 128, and more complex rules :

— Rule 3 : $(3)_{10} = (11)_2$, $3 = 1 + 2$,

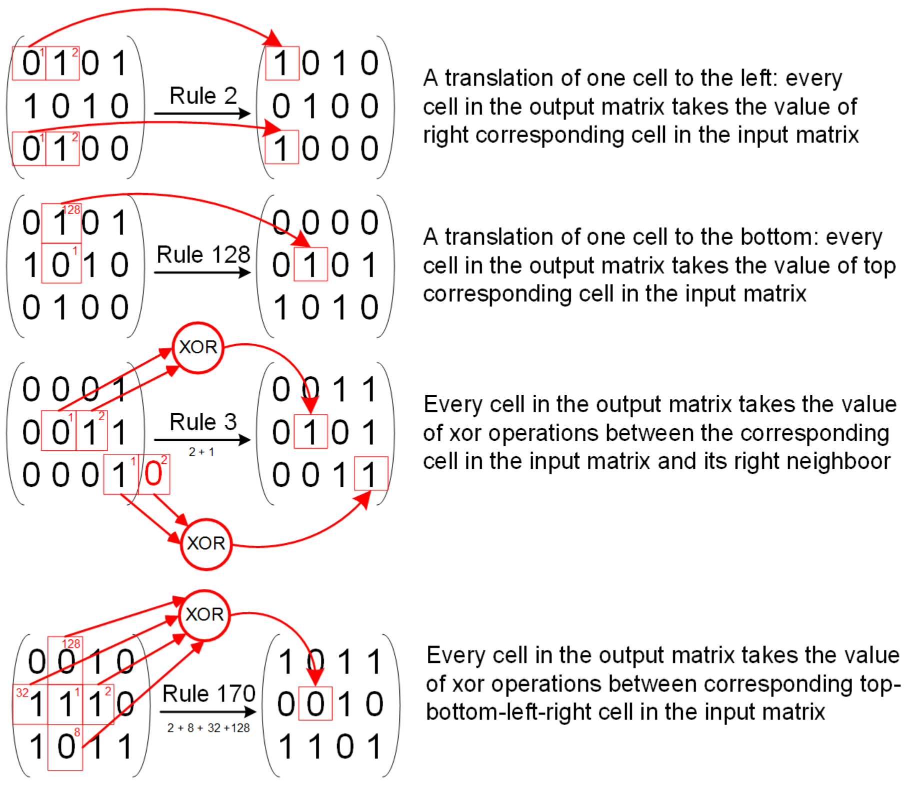
— Rule 170 : $(170)_{10} = (10101010)_2$, $170 = 2 + 8 + 32 + 128$,

Each rule is applied uniformly to each cell of a problem matrix of order (3 x 4) with null boundary condition (extreme cells are connected with logic-0 states).

Fundamental rules namely rule 2, 4, 8, 16, 32, 64, 128, 256 cause translation of the image/matrix in the directions mentioned against each rule in the table 2.

The extent of shift is dependent on the number of repetitions of the application of such rules as illustrated in the following figures 6 and 7.

1. XOR : `https://en.wikipedia.org/wiki/Exclusive_or`



Rule 2
A translation of one cell to the left: every cell in the output matrix takes the value of right corresponding cell in the input matrix
Rule 128
A translation of one cell to the bottom: every cell in the output matrix takes the value of top corresponding cell in the input matrix
XOR
Rule 3
2 + 1
XOR
Every cell in the output matrix takes the value of xor operations between the corresponding cell in the input matrix and its right neighboor
XOR
Rule 170
2 + 8 + 32 +128
Every cell in the output matrix takes the value of xor operations between corresponding top-bottom-left-right cell in the input matrix

FIGURE 5 – Illustration of the application of rules 2, 128, 3 and 170.

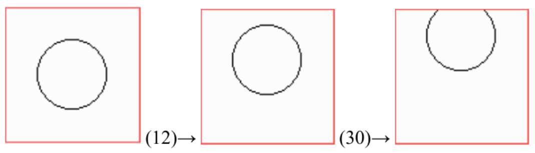


FIGURE 6 – Application of rule 8, 12 times, and 30 times to a given input matrix representing a circle.

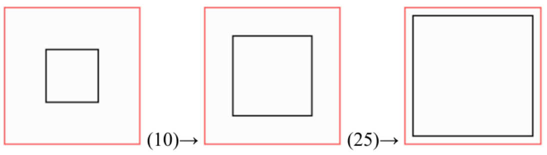


FIGURE 7 – Successive applications of rule 2, 32, 8, and 128, 10 times, and 25 times to a given input matrix representing a square.

4 Work to achieve

4.1 Library

A library written in C containing the types and functions described above. This library will be implemented using at least two files in C :

matrix.h the header file which contains the prototype of the functions, types, constants and variables provided by the library ;

matrix.c the associated source file which contains the body of the various proposed functions.

You must also provide :

Makefile makefile to compile the sources, generate libraries and executable. At least, 3 targets must be defined in this Makefile :

- **all** compiles everything, generates the libraries and executable.
- **lib** generates the binary code of the matrix libraries *libMatrix.so*.
- **clean** cleans the tmp files generated during the compilation process and the various binary files

matrixmain.c the main program, which contains the core of the program and its associated GUI.

The compilation of the entire project have to be managed thanks to a **Makefile**. The matrix dynamic library may be compiled and produced using the following command line (in the Makefile) :
$gcc -Wall -Werror -ansi -pedantic <source files> -o libMatrix.so -shared -fpic

4.2 Representation of matrices in C

Figure 8 describes the linked-list based structure used to represent a 2×2 matrix in this project. This structure is based on two doubly linked-list that index each row and column of the matrix. Each cell of the matrix is linked to the next cell on the same row, and to the next cell on the same column. The entire cell structure leads to the creation of a kind of grid structure.

Constraints :
— The considered matrices are not necessarily square ;
— We are only considering two-dimensional matrix
— These matrices are storing boolean values ;

To define the matrix structure, you have at least to provide in C the definition of the following data types :

Matrix
Matrix corresponds to the type of the entire matrix structure, it is composed of four different fields :

- **colCount : integer** the number of columns of this matrix ;
- **rowCount : integer** the number of rows of this matrix ;
- **cols : pointer on colElement** a pointer on the first column element ;
- **rows : pointer on rowElement** a pointer on the first row element.

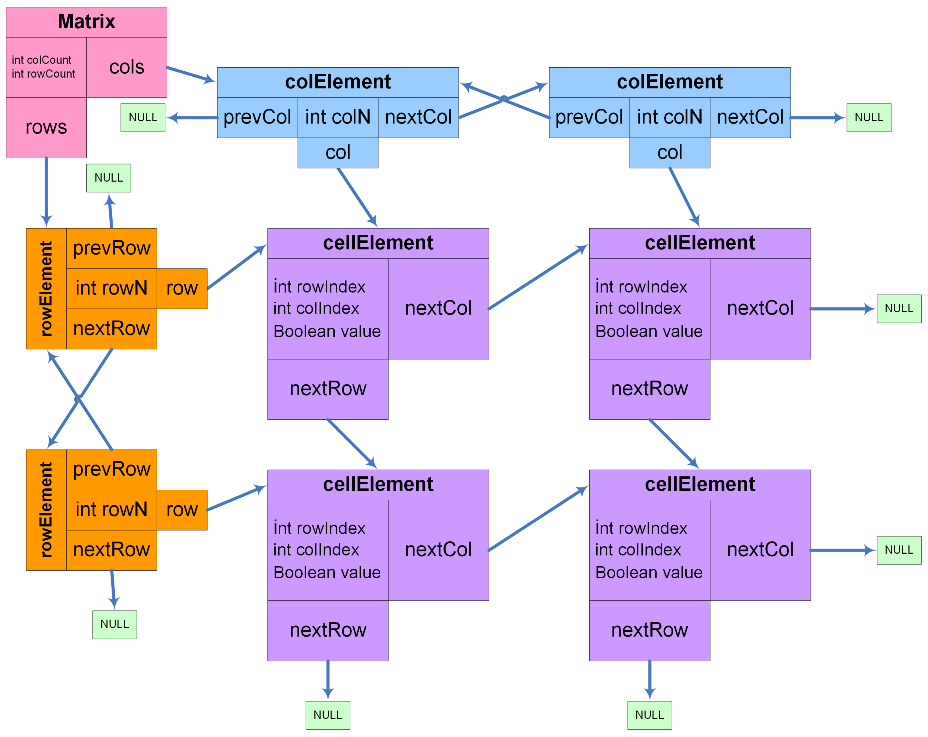


FIGURE 8 – Example of 2×2 matrix represented using simply/doubly linked lists

colElement
An element which represents one column inside the matrix, it is composed of four different fields :

colN :integer the index of the corresponding column inside the matrix ;

nextCol : pointer on colElement a pointer on the next column of the matrix ;

prevCol : pointer on colElement a pointer on the previous column of the matrix ;

col : pointer on cellElement a pointer on the first cell of the corresponding column inside the matrix.

rowElement
An element which represents one row inside the matrix, it is composed of four different fields :

rowN : integer the index of the corresponding row inside the matrix ;

nextRow : pointer on rowElement a pointer on the next row of the matrix ;

prevRow : pointer on rowElement a pointer on the previous row of the matrix ;

row : pointer on cellElement a pointer on the first cell of the corresponding row inside the matrix.

cellElement
An element which represents one cell inside the matrix, it is composed of five different fields :

colIndex : Integer the column index of the corresponding cell inside the matrix ;

rowIndex : Integer the row index of the corresponding cell inside the matrix ;

value : Boolean the boolean value of the corresponding cell inside the matrix ;

nextCol : pointer on cellElement a pointer on the next cell in the corresponding column of the matrix ;

nextRow : pointer on cellElement a pointer on the next cell in the corresponding row of the matrix ;

4.3 Main program and graphical user interface

To provide a main program using the previous library and enabling a user to test all of its provided functionalities in a simple and friendly way.

Since your program should enable to simply test each library functions, it must provide a random generation mechanism of the various matrices required for each operation.

This program may be compiled using the following command line (in the Makefile) :

$ gcc -Wall -Werror -ansi -pedantic -L<library directory> MatrixMain.c -o Matrix.exe -lMatrix

In running the program, the variable LD_LIBRARY_PATH have to specify the directory containing the *Matrix* library.

5 The project deliverables

A ZIP archive or TAR.GZ containing

- the report in pdf format describing every algorithms (**making no assumptions about the considered list implementation**) and the corresponding results.
- src/
  - Makefile (Linux compliant) to build the libraries and get the executable Matrix.exe associated to the file matrixmain.c
  - source files of the libraries : matrix.c, matrix.h and all other files containing the definition of the other required types.
  - the binary code of the library *libMatrix.so*
  - the main program matrixmain.c

6 BONUS question I

Doing the same thing but without storing zero values in the matrix data structure. The aim is to save memory by storing only the values that are really necessary.