Durand Lilian
Poncet Yann

02/01/2017

# LO27 Project Report

## Table des matières



## Introduction

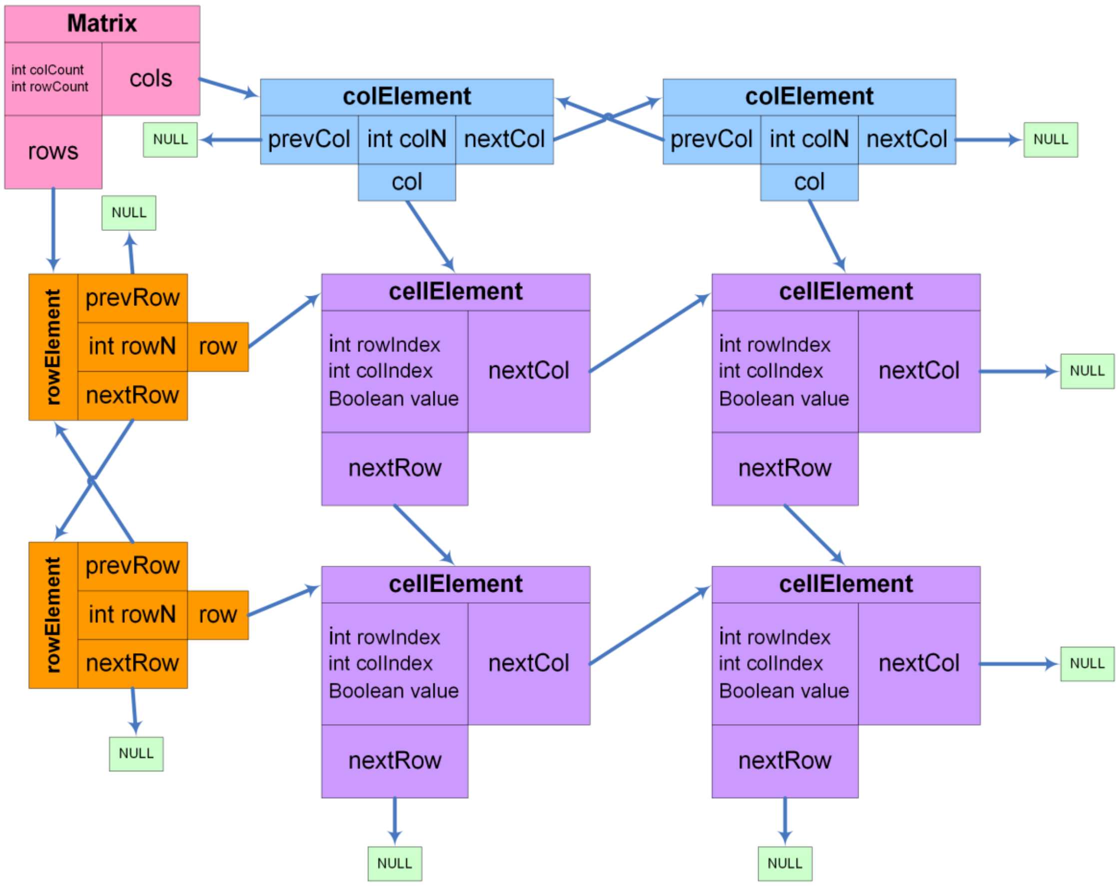
The goal of this project aims at providing the definition of a new abstract data type called Matrix and the set of associated functions to manipulate this new type. In the report, all the algorithms of these functions are described. The objective in terms of C code consist in providing a library of features for handling matrices (the source codes are in a file in the same folder).

In the report we will first describe the new abstract data type Matrix and each abstract data type used with the Matrix data type. We will also provide a C representation of these new abstract data type. Then we will provide all the algorithms describing each function useful to carry out the project as well as the requested functions. Finally there will be a conclusion dealing with an evaluation of our personal work, a summarization of our work and the introduction of some optimizations.

Durand Lilian
Poncet Yann
02/01/2017

# Description of the new abstract data types

## Representation of the data types




## Boolean

The Boolean data type is used all along the project because the Matrix is filled with Booleans. It can take two values: TRUE (1) or FALSE (0).

C representation:

```
#define Boolean int
#define TRUE 1
#define FALSE 0
```

## cellElement

The CellElement data type is at the lowest level. It is an element of a linked list containing the Boolean, a row index and a column index. It is linked with the cell in the next row and the cell in the next column (thanks to pointers).

C representation:

```
typedef struct cellElement cellElement;

struct cellElement
{
	int colIndex;
	int rowIndex;
	Boolean value;
	cellElement* nextCol;
	cellElement* nextRow;
};
```

### colElement and rowElement

As we can see in the diagram, the ColElement and the RowElement are elements of doubly linked list that allows us to access to the cell that have the corresponding index. Each of those elements contains an index and allows us to access to the next or previous row (respectively column) and the cell having the same row index and a column index equal to 1 (respectively column and row).

C representation:

```
struct colElement
{
	int colN;
	colElement* nextCol;
	colElement* prevCol;
	cellElement* col;
};

typedef struct rowElement rowElement;

struct rowElement
{
	int rowN;
	rowElement* nextRow;
	rowElement* prevRow;
	cellElement* row;
};
```

### Matrix

Matrix is the final data type, a structure containing the number of rows and of columns as well as a pointer on the first colElement and on the first rowElement.

C representation:

```
typedef struct
{
	int colCount;
	int rowCount;
	colElement* cols;
	rowElement* rows;
}Matrix;
```

Durand Lilian
Poncet Yann 02/01/2017

# Algorithms of some functions

## Function searchCell

Documentation

matrix : the matrix in which we want to find the cell
rowV : an integer, the index of the cell's row we want to find
colV : an integer, the index of the cell's column we want to find
rowE : a rowElement used to navigate in the matrix
cell : the cellElement we are looking for

```
Function searchCell(matrix : Matrix, rowV : Integer, colV : Integer) : cellElement
BEGIN
    if(isMatrixEmpty(matrix)) then
        Write "Error, the matrix is empty"
        xorMatrix ← EMPTY
    end if

    rowE ← rows(matrix)

    while(rowN(rowE) ≠ rowV) do
        rowE ← nextRow(rowE)
    done

    cell ← row(rowE)
    i ← 0

    for i from 1 to colV do
        cell ← nextCol(cell)
    done

    searchCell ← cell
END
```

## Function newMatrix

Documentation

b : a double array of booleans, the representation with an array of the desired matrix
nbCol : an integer, the number of columns in the desired matrix
nbRow : an integer, the number of rows in the desired matrix
i,j,k : integers used in loops
matrix : a Matrix, the matrix that we want to build here
colE : a colElement, used to create the colElements of the desired matrix
rowE : a rowElement, used to create the rowElements of the desired matrix
new : a cellElement, used to create the cells of the desired matrix
new2 : a cellElement, used to create the cells of the desired matrix (in another loop)

```
Function newMatrix(b : Boolean [0 … nbRow-1][0 … nbCol-1], nbCol : Integer, nbRow : Integer) :
Matrix
BEGIN
        i ← 0
        j ← 0
        k ← 0
        colCount(matrix) ← nbCol
        rowCount(matrix) ← nbRow

        colN(colE) ← 1
        prevCol(colE) ← EMPTY

        rowN(rowE) ← 1
        prevRow(rowE) ← EMPTY

        cols(matrix) ← colE
        rows(matrix) ← rowE

        for i from 1 to nbCol do
                nextCol(colE) ← new
                prevCol(new) ← colE
                colN(new) ← i+1
                colE ← new
        done
        nextCol(colE) ← EMPTY

        for j from 1 to nbRow do
                nextRow(rowE) ← new
                prevRow(new) ← rowE
                rowN(new) ← j+1
                rowE ← new
        done
        nextRow(rowE) ← EMPTY
```

```
        cell ← EMPTY
        for i from nbRow-1 to -1 do                    (starting at the end of the matrix)
        |       for j from nbCol-1 to -1 do
        |       |       nextCol(new) ← cell
        |       |       value(new) ← b[i][j]
        |       |       colIndex(new) ← i+1
        |       |       rowIndex(new) ← j+1
        |       |       if nextRow(rowE) = EMPTY then
        |       |       |       nextRow(new) ← EMPTY
        |       |       else
        |       |       |       rowE ← nextRow(rowE)
        |       |       |       new2 ← EMPTY
        |       |       |       new2 ← row(rowE)
        |       |       |       for k from 0 to j do
        |       |       |       |
        |       |       |       done
        |       |       |       nextRow(new) ← new2
        |       |       |       rowE ← prevRow(rowE)
        |       |       end if
        |       |       cell ← new
        |       |       row(rowE) ← cell
        |       |
        |       |       col(colE) ← cell
        |       |       colE ← prevCol(colE)
        |       |
        |       done
        |       cell ← EMPTY
        |       colE ← cols(matrix)
        |       for k from 1 to nbCol do
        |       |       colE ← nextCol(colE)
        |       done
        |       rowE ← prevRow(rowE)
        done
        newMatrix ← matrix
END
```

## Function applyRules

Documentation

matrix : the matrix we will use to apply the rules
ruleID : an integer, what rule we want to use
loop : an integer, the number of time we want to apply the rule
resultMatrix : the resulting Matrix

```
Function applyRules(matrix : Matrix, ruleID : Integer, loop : Integer) : Matrix
BEGIN
        if(isMatrixEmpty(matrix)) then
                Write "Error, the matrix is empty"
                xorMatrix ← EMPTY
        end if

        resultMatrix ← matrix

        if(loop>0) then
                resultMatrix ← applyRules(rulesDecomposition(matrix, ruleID, 1),
                ruleID, loop-1)
        end if

        applyRules ← resultMatrix
END
```

Durand Lilian
Poncet Yann 02/01/2017

## Function xorMatrix

Documentation

matrix1, matrix2 : the two Matrix we want to merge with a XOR
matrix3 : the resulting Matrix once the XOR is applied
i,j : integers used in loops
b : a double array of Booleans, used to create the resulting Matrix (thanks to newMatrix)

```
Function xorMatrix(matrix1 : Matrix, matrix2 : Matrix) : Matrix
BEGIN
        i ← 0
        j ← 0

        if ((colCount(matrix1) ≠ colCount(matrix2) OR rowCount(matrix1) ≠ rowCount(matrix2))
        OR (isMatrixEmpty(matrix1) OR isMatrixEmpty(matrix2))) then
                Write "Error, the matrix is empty"
                xorMatrix ← EMPTY
        end if

        for i from 0 to rowCount(matrix1) do
                for j from 0 to colCount(matrix1) do
                        b[i][j] ← abs(value(searchCell(matrix1,i+1,j+1)) -
                        value(searchCell(matrix2,i+1,j+1)))
                done
        done

        Matrix matrix3 ← newMatrix(b,colCount(matrix1),rowCount(matrix1))

        xorMatrix ←
        matrix3
END
```

## Function newVanishingArray

<u>Documentation</u>

nbCol : an integer, the number of columns we want in the array of boolean
nbCol : an integer, the number of columns we want in the array of boolean
b : a double array of integer, the result
i and j : integers

```
Function newVannishingArray(nbCol : Integer, nbRow : Integer) : Boolean[0 … nbRow-1][0 … nbCol-1]
BEGIN
        i ← 0
        j ← 0
        for i from 0 to nbRow do
                for j from 0 to nbCol do
                        b[i][j] ← 0
                done
        done

        newVannishingArray ← b
END
```

## Functions rule

Documentation

matrix : the Matrix we want to apply the rule to
i,j : integers used for the loops
b : a double array of Booleans used to create the resulting Matrix (once the rule is applied to the array)
resultMatrix : the resulting Matrix (once the rule is applied)

```
Function rule2(matrix : Matrix) : Matrix
BEGIN
        if(isMatrixEmpty(matrix)) then
                Write "Error, the matrix is empty"
                xorMatrix ← EMPTY
        end if

        i ←0
        j ← 0

        for i from 0 to rowCount(matrix) do
                for j from 0 to colCount(matrix) do
                        b[i][j] ← value(searchCell(matrix,i+1,j+2))
                done
                b[i][colCount(matrix)-1] ← 0
        done

        resultMatrix ← newMatrix(b, colCount(matrix), rowCount(matrix))

        rule2 ←
        resultMatrix
END

Function rule8(matrix : Matrix) : Matrix
BEGIN
        if(isMatrixEmpty(matrix)) then
                Write "Error, the matrix is empty"
                xorMatrix ← EMPTY
        end if

        i ←0
        j ← 0

        for i from 0 to rowCount(matrix) do
                for j from 0 to colCount(matrix) do
                        if(i = rowCount(matrix)-1) then
                                b[i][j]=0
```

```
                        else
                                    b[i][j] =
                                    value(searchCell(matrix,i+2,j+1))
                        end if
                done

        done

        resultMatrix ← newMatrix(b, colCount(matrix), rowCount(matrix))

        rule8 ← resultMatrix
END
```

Function rule32(matrix : Matrix) : Matrix

```
BEGIN
        if(isMatrixEmpty(matrix)) then
                Write "Error, the matrix is empty"
                xorMatrix ← EMPTY
        end if

        i ←0
        j ← 0

        for i from 0 to rowCount(matrix) do

                b[i][0] ← 0
                for j from 1 to colCount(matrix) do
                            b[i][j] ←
                            value(searchCell(matrix,i+1,j))
                done
        done

        resultMatrix ← newMatrix(b, colCount(matrix), rowCount(matrix))

        rule32 ← resultMatrix
END
```

Function rule128(matrix : Matrix) : Matrix

```
BEGIN
        if(isMatrixEmpty(matrix)) then
                Write "Error, the matrix is empty"
                xorMatrix ← EMPTY
        end if

        i ←0
        j ← 0
```

```
for i from 0 to rowCount(matrix) do
    for j from 1 to colCount(matrix) do
        if(i=0) then
            b[i][j] ← 0
        else
            b[i][j] ←
            value(searchCell(matrix,i,j+1))
        end if
    done
done

resultMatrix ← newMatrix(b, colCount(matrix), rowCount(matrix))

rule128 ←
resultMatrix
END

Function rule4(matrix : Matrix) : Matrix
BEGIN
        if(isMatrixEmpty(matrix)) then
                Write "Error, the matrix is
                empty"
                xorMatrix ← EMPTY
        end if

        resultMatrix ← rule8(rule2(matrix))

        rule4 ← resultMatrix
END

Function rule16(matrix : Matrix) : Matrix
BEGIN
        if(isMatrixEmpty(matrix)) then
                Write "Error, the matrix is empty"
                xorMatrix ←
                EMPTY
        end if

        resultMatrix ← rule8(rule32(matrix))

        rule16 ← resultMatrix
END
```

```
Function rule256(matrix : Matrix) : Matrix
BEGIN
        if(isMatrixEmpty(matrix)) then
                Write "Error, the matrix is empty"
                xorMatrix ← EMPTY
        end if

        resultMatrix ← rule128(rule2(matrix))

        rule256 ← resultMatrix
END

Function rule64(matrix : Matrix) : Matrix
BEGIN
        if(isMatrixEmpty(matrix)) then
                Write "Error, the matrix is empty"
                xorMatrix ← EMPTY
        end if

        resultMatrix ← rule128(rule32(matrix))

        rule64 ← resultMatrix
END
```

## Function rulesDecomposition

Documentation

matrix : the Matrix we want to apply the rules to
ruleID : an integer, the rule we want to apply
bit : an integer, telling in which bit we will work (power of 2)
tmpMatrix : a Matrix used to store the matrix created after a rule
resultMatrix : the resulting Matrix once all the rules are applied (after the recursivity)

```
Function rulesDecomposition(matrix : Matrix, ruleID : Integer, bit : Integer) : Matrix
BEGIN
    if(isMatrixEmpty(matrix)) then
        Write "Error, the matrix is empty"
        xorMatrix ← EMPTY
    end if

    resultMatrix ← newMatrix(newVannishingArray(colCount(matrix),
    rowCount(matrix)), colCount(matrix), rowCount(matrix))

    if(ruleID>0) then

        if((ruleID & 1) = 1) then

            Choose bit among

                1 : tmpMatrix ← matrix
                2 : tmpMatrix ← rule2(matrix)
                4 : tmpMatrix ← rule4(matrix)
                8 : tmpMatrix← rule8(matrix)
                16 : tmpMatrix ← rule16(matrix)
                32 : tmpMatrix ← rule32(matrix)
                64 : tmpMatrix ← rule64(matrix)
                128 : tmpMatrix ←
                rule128(matrix)
                256 : tmpMatrix ←
                rule256(matrix)
            end

            resultMatrix ← xorMatrix(tmpMatrix,
            rulesDecomposition(matrix, ruleID>>1, bit*2))

        else

            resultMatrix ← rulesDecomposition(matrix,
            ruleID>>1, bit * 2)
        end if
```

```
        end if

        rulesDecomposition ← resultMatrix
END
```

## Other information about our project

To build a new matrix quickly we made a function called newAleaArray. Namely that the size of the random array is defined in the .h code.

We had first some trouble making the function “newMatrix” mostly because of the pointers on NULL (and because we began to fill the Matrix starting at the beginning). After a few thoughts we decided to fill the Matrix starting at the end and so it was really easily and fast to fill it.

When we started to think about the function printMatrix, we thought that it would be better to have a function capable of finding a cell knowing its index in the Matrix that’s why we implemented a function called searchCell which allow us to do this.

The real challenge was the function applyRules. After some research about the binary in C we found the operator ">>" which help us to binary analyze the rule and for each bit equal to 1 the corresponding rule will be applied thanks to the weight of the corresponding bit (the variable bit in the function rulesDecompostion is the value of the weight corresponding). Then a XOR will be applied between all the returned Matrix thanks to recursivity to create the resulting matrix. Then if we want to apply this rule several times, the process will start again and merge all the resulting matrix to have the final matrix.

## Conclusion

This project was a great opportunity to apply our knowledge to a concrete issue, to learn new C possibilities (especially with the function applyRules) and to deal with a deadline knowing that we would have trouble to see each other to merge our work. Some of the optimizations that we introduced are described right above the conclusion.