Programming Assignment

We are going to look at the examples of simple programs or what is best known as cellular automata. These entities are studied extensively in Stephen Wolfram’s book “New Kind of Science” <http://www.wolframscience.com/nksonline/toc.html>

In brief, we have a matrix with N columns and M rows. The choice of internal storage (row-first or column-first) is up to you. The print-out is expected in the following format. We start with 1st row which is at the top. Row 2 is right below Row 1. Row 3 is below Row 2, and so on. Each entry in the matrix is either 0 or 1. Let define each matrix element as A[i][j]. Here index i runs between 0 and N-1, index j runs between 0 and M-1.

We always initialize Row 1. It is our initial condition. Given a rule (we discuss this later), we construct Row 2 by applying the rule to the 1st row. Row 3 is constructed by applying the same rule to Row 2, and so on. We can extend the number of rows indefinitely. We pay special attention to boundaries of each row: first two and last two elements of each row. We only do not need to apply boundary conditions to Row 1. For the rest of rows, the boundary conditions are set as follows: A[0][j] == A[N-2][j] and A[1][j] == A[N-1][j].

Now lets describe rules of cellular automata. We focus on rules that define the element value A[i][j] based on three elements: A[i-1][j-1], A[i][j-1], A[i+1][j-1]. Wolfram has studied many such rules. We focus on his “Rule 30” that is graphically represented on page 53 of his book: <http://www.wolframscience.com/nksonline/page-53> On the picture at the top of page 53, rule 30 is in lower left quadrangle. Elements with value 1 are dark, elements with value 0 are white. If we represent values A[i-1][j-1], A[i][j-1], A[i+1][j-1] as a tuple with 3 values, we can summarize Rule 30 for value A[i][j] as follows

|  |  |
| --- | --- |
| (1,1,1) | 0 |
| (1,1,0) | 0 |
| (1,0,1) | 0 |
| (1,0,0) | 1 |
| (0,1,1) | 1 |
| (0,1,0) | 1 |
| (0,0,1) | 1 |
| (0,0,0) | 0 |

*Now, here is what we want you to develop.*

Develop a fully working code that runs from a command line. You provide the source code which should be contained in a single file. Please use either Native C++ (I will compile it with Visual Studio) or Python (v 2.7). You can use standard libraries (STL/Boost are allowed). No snippets of codes from 3rd party places are allowed. So please do not use libraries that implement cellular automata. Your code expects one input file in the same directory (first\_row.csv). This csv file contains only values (no headers) for the first row e.g. 0,1,0,0,0,1,0. No need for any additional information. Your code prints to screen 100 rows (including the first row). Print statement contains only 1 or 0, e.g. 000000100000100000.

Your code must contain a class fully specifying Rule 30. Call this class Rule30Class. It’s instance is given as instance\_of\_rule30. Your code must contain function new\_row that returns a row object given previous row, the rule, and another object that I will describe below):

next\_row=new\_row(instance\_of\_rule30, prev\_row, pattern\_map) (if C++) or

next\_row,pattern\_map=new\_row(instance\_of\_rule30, prev\_row, pattern\_map) (if Python)

Your code updates the locations (map) of unique patterns for two rows, given by pattern\_map. The pattern is based on 6 matrix elements (denoted as “six-value pattern”) located in two subsequent rows:

|  |  |  |
| --- | --- | --- |
| A[i-1][j-1] | A[i][j-1] | A[i+1][j-1] |
| A[i-1][j] | A[i][j] | A[i+1][j] |

For example, pattern 1-1-0/1-0-0 is given fully specified as this:

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 0 |
| 1 | 0 | 0 |

As you can imagine, there can be multiple occurrences of six-value pattern in the matrix. We want to know all locations where such patterns occurs. Instance pattern\_map of some convenient class contains mapping from six-value pattern to list of coordinates. It is updated during the construction of all rows. After you process 99 new rows and print their values, you must call another function get\_pattern\_occurrencies(pattern\_k, pattern\_map) three times (k=1,2,3). Here pattern\_k is an instance of another class SixValuePattern. Calls to function get\_pattern\_occurrencies are fast, no loops through the matrix must be done during this call. Instance of SixValuePattern is initialized to any combination of six values. Function get\_pattern\_occurrencies returns a full list of coordinates Xi, Yj that represent the location of a central element in the second row (shaded in the table above). Coordinates Xi, Yj are matrix indices (column i and row j). Your function should be general enough, but please print (to the screen) all occurrences of these three patterns (k=1,2,3): 1-1-0/1-0-0, 1-1-0/1-0-1, 1-1-0/1-1-0

That is all. Good luck!