CSE 6010 Assignment 1: Modeling Urban Segregation

Due Dates:

- Due: 11:00 AM, Friday, September 8, 2017
- Revision (optional): 11:55 PM, Monday September 11, 2017

An economist named Thomas Schelling developed what is now a well-known model characterizing the creation of segregated neighborhoods in cities. The goal of this assignment is to write a simulation program in C to experiment with Schelling's model, and report what your experiments say regarding the development of segregated communities based on the simplified assumptions used in this model.

The city is divided into a checkerboard-like two-dimensional grid. Assume there are two types of citizens living in the city of different ethnic background – red and green citizens. A square (cell) in the checkerboard is in one of three states: red if it is occupied by a red citizen, green if occupied by a green citizen and white if the cell is vacant. A neighborhood is defined as the (up to eight) cells adjacent to a particular cell. Note that cells along an edge or the corner of the city have smaller neighborhoods.

Red citizens have a preference to have a least some fraction *f* of neighbors who are also red, and the same for green citizens. For example, if *f* is 30%, and a red citizen has 6 neighboring cells that are occupied (and 2 are vacant), this individual will be satisfied if at least 2 (30% of 6 or 1.8) of these 6 neighbors are also red, and dissatisfied otherwise.

The simulation operates as follows:

1. The checkerboard is initialized so each cell has a probability of 0.40 to be red, 0.40 to be green, and 0.20 to be vacant.
2. A scan is made through all occupied cells to mark those cells that hold a dissatisfied citizen.
3. For each dissatisfied cell: the person in that cell moves to another cell. This cell is selected by first randomly selecting any cell in the city (independent of its state), and then searching from this initial selection for the first available cell that is (1) vacant, and (2) in a state where this individual will be satisfied if moved there, ignoring the plans of individuals who are trying to move. The individual then moves to this cell, marking the cell it left as vacant. The search for a cell where the individual will be satisfied should start from the randomly selected cell, and proceed from left to right, and down row by row; if the cell in the bottommost row and rightmost column is reached and found to be unsatisfactory, the search continues with the cell at the top left corner of the grid. If no cell can be found where the individual will be satisfied, the individual leaves the city, leaving the cell it had occupied vacant. Note that during this step an individual's move is based on the current state of the grid, including other moves that have already been processed during this time step.
4. The simulation advances to the next time step, and repeats steps 2 and 3 above.

The simulation completes when either (1) all individuals are satisfied, or (2) 100 time steps have been reached.

The model described above is referred to as a cellular automaton. Cellular automata are widely used in the sciences, engineering, and social sciences to model a variety of different types of systems and phenomena.

To complete this assignment, develop a simulation of the system described above using the C programming language. Simulate a 100x100 grid. However, your program should be written so that the grid size can be easily changed. You are provided with a template source code that details the structure of the program. **The parts you need to implement are marked by a TODO statement.** You should not modify any of the function prototypes already provided, but you can definitely add some more to help you organize your code even further. You can get information about how the program can be run by simply running your executable without any command-line inputs. Specifically:

1. Implement the function with the prototype

   ```
   char **malloc_matrix(int n1, int n2);
   ```

   This function creates a two-dimensional matrix using dynamic memory allocation (use the built-in C function `malloc()` for this purpose) and returns a pointer to the structure that was created. The created matrix has n1 rows and n2 columns. Each element of the matrix holds a character (`R`, `G`, or `' '` for red, green, and vacant, respectively). The matrix must be structured using the *array of pointers* representation, i.e., the data for the two-dimensional matrix is stored in a set of one-dimensional arrays, one array for each row of the matrix, and an additional, separate one-dimensional array with each array element containing a pointer to the data for a single row. The function returns `NULL` if the operation failed, e.g., due to insufficient memory.
2. Implement the function with the prototype

   ```
   void free_matrix (int n1, int n2, char **a);
   ```

   Release the memory allocated for the matrix pointed to by `a` which has `n1` rows and `n2` columns. Use the built-in C function `free()` to release a block of memory.
3. Implement the function with the prototype

   ```
   void init_matrix(int n1, int n2, char **a);
   ```

   that initializes each cell of the matrix as described above.
4. Implement the function with the prototype

   ```
   int simulate (char **a, int n1, int n2, double f);
   ```

   Simulate one time step using the matrix pointed to by `a` with `n1` rows and `n2` columns, where `f` determines the minimum number of desired neighbors similar to the occupant, as described above. This function returns an integer indicating the number of citizens that moved, or left the game in this time step. It returns a negative number if an error occurred in the simulation, e.g., an invalid value for `f` is specified.

5. Implement the function with the prototype

   ```
   void print_matrix(int n1, int n2, char **a);
   ```

that prints the matrix `a` in a readable format (i.e., columns are aligned).

Conduct a literature search and write up a brief description characterizing what you anticipate will be the results of simulations using this model. Then complete a set of simulation runs using different values of `f` to determine if your results are consistent with these expectations. Do your results agree with other results reported in the literature? Explain why or why not. Are there other applications where these results are relevant? While we are not expecting a fully comprehensive literature search, you should aim for a page or two of text with perhaps four or five related works cited. Your report should include print outs of the initial and final states of representative simulation run(s) to substantiate the discussion in your report.

Turn your report and software in as a single zip file. Your software must be well documented and include comments so the code is easy to understand. You should include a README file with instructions on how to compile and run your program.

The web is an excellent resource to answer specific questions on C. We encourage you to use it, but be careful not to utilize code copied directly from the web.

A reminder you must adhere to the Georgia Tech honor code, and the collaboration policy stated in the course syllabus. Specifically, you are encouraged to discuss the problem and possible solutions with other students (as well as the TA/instructor), however, all code that you turn in must be completely your own work. Disseminating your code to other students is strictly prohibited. Further, downloading code from the web or other sources other than examples provided in class is also prohibited.