EECS 592
FOUNDATIONS OF ARTIFICIAL INTELLIGENCE

Issued: Jan. 15, 2020
Due: Mon., Jan. 27, 2020 (11:59pm to Canvas)

PROGRAMMING REVIEW ASSIGNMENT

Please submit all files required by this assignment to Canvas. We ask that you use one programming language for all problems in this assignment: either Python or C++, but not both. Please take care to complete this assignment independently. Do not use any code found online or written by other students. If you are struggling please go to our office hours and/or email.

1. **Logic:** A cellular automaton (CA) consists of a collection of cells which can hold data values, generally organized into some regular geometric arrangement (usually a square grid or a line). A one-dimensional CA (1dCA) has all the cells arranged in a line. The values in the cells change over time. Each cell's value at the next time interval depends on its current value as well as the value of its two immediate neighbors. We set the first and last cells to be always empty. In this case you will be implementing a 1dCA with two states (full and empty, which you should implement as 1 and 0).

   The 1dCA you will implement has the following update rule:

   1. If the current cell is empty, and one of the two adjacent cells is full (but not both), set the current cell to be full.
   2. Otherwise set the current cell to be empty.

   You have to iterate through all the cells to generate the new pattern (the next "generation"). You need to have two arrays to store the data, because you are not allowed to modify one generation until the next generation has been completely computed. The 1dCA should be randomly seeded with 1's and 0's initially (e.g., about half and half, subject to random number generation for each program execution). If successive generations are printed on a screen, one generation per line, with (say) a "." for 0s and a "*" for 1s, you will get a chaotic-looking pattern that has a fractal-like quality.

   Write a python script (or a C++ program) to implement a cellular automaton. Your script or program should prompt the user to enter the numbers of cells in the 1dCA and the number of generations to compute. The script/program should then print this many generations to the screen. Dynamically create the arrays or Python data structures you need to hold the 1dCA data. You are free to design your own internal function(s) as you'd like, but please try to make your code clear, efficient, and understandable to our grader. Submit your code for this problem to Canvas as `cell.py` or `cell.cpp`.

2. **Array Sorting:** The Quicksort algorithm is a popular method to sort databases into an ordered list. Most of you have likely encountered the quicksort algorithm before, but if not, you can find a description of it on https://en.wikipedia.org/wiki/Quicksort using the Lomuto partition scheme. Write and submit to Canvas a script (or program) `quicksort.py` (or `quicksort.cpp`) that implements the Quicksort algorithm. Your script/program should read file `unsorted.txt` and write file `sorted.txt` as output. Within your script/program, write a distinct function `quicksort()` that takes as input the unsorted array (or array pointer) and returns a sorted array (or array pointer). The script or `main()` C++ program should handle file input/output;

only the data structures are known to your quicksort function. We will evaluate your code by executing it on a potentially large `unsorted.txt` file formatted with one database entry per line. Each line will be formatted in a CSV (comma-separated variable) format, as shown by example below. Sort the array numerically by the student's homework grade. Make your `quicksort()` function as general as possible, and DO NOT use any functions (e.g., `qsort`) built into the Python or C++ languages. We will not grade your code on its generality, but if you find this assignment straightforward please challenge yourself to make your quicksort() function work for any "key" in your database (e.g., last name, first name, or homework grade). We will only be running your primary script / program so input/output arguments for your `quicksort()` function are up to you.

Sample `unsorted.txt`:
Atkins, Ella, 94
Krishnan, Abhiram, 100
…

3. **Graph Construction** (Preparation for the next assignment.) Create a graph for the following road map used in the Russell & Norvig text. The file, `graph.txt,` has the following CSV format:

```
Arad, Zerind, 75
Arad, Sibiu, 140
Arad, Timisoara, 118
Bucharest, Pitesti, 101
Bucharest, Urziceni, 85
Bucharest, Giurglu, 90
Craiova, Dobreta, 120
Fagaras, Bucharest, 211
Hirsova, Eforie, 86
Lasi, Neamt, 87
Lugoj, Mehadia, 70
Mehadia, Dobreta, 75
Oradea, Sibiu, 151
Pitesti, Craiova, 138
Rimnicu Vilcea, Pitesti, 97
Rimnicu Vilcea, Craiova, 146
Sibiu, Fagaras, 99
Sibiu, Rimnicu Vilcea, 80
Timisoara, Lugoj, 111
Urziceni, Hirsova, 98
Urziceni, Vaslui, 142
Vaslui, Lasi, 92
Zerind, Oradea, 71
```

Write and submit to Canvas a script `mygraph.py` or program `mygraph.cpp` to create a graph data structure and print the number of vertices (nodes) and edges to the screen. You can test your code on the above graph; we will grade your code on a different graph in the same format. You can reuse your code in future projects, e.g., for search.

OPTIONAL (not graded): Add to the end of your code (easiest in Python) a graphical interface that shows your graph on the screen. [This is a convenience for you – it is not required but it is cool, and early use of graphics will make it easier to use graphics later!]