Project 2: Contagion

Instructions

For this project, you and your group will be designing and implementing a system in C or C++, that will act as a simulation engine to compare the impact of different vaccine distribution strategies on the spread of a disease. Specifically, you will be examining how prioritizing vaccine distribution across areas using closeness centrality and degree centrality compare against each other and random and equal distribution strategies. The results of these experiments should be provided to the user at the end in an easy to read format.

Additionally, sample input files can be found on the CSE machines at `/home/dmk0080/2110/proj2/` You can copy input the files from there.

Further, you will need to utilize the GitLab code repository located at https://apollo.cse.unt.edu/ . You may access it from the website, the terminal, or an IDE of your choice.

Also, as a reminder, all of the work for this project must be the sole product of the group. You may not share code with other groups, or download solutions off the internet, as doing so will be considered cheating. If you are struggling with a concept or implementation, you are encouraged to contact the instructor or your TA's for aid.

Requirements

This assignment has two parts: a wiki portion and an implementation portion.

Wiki

For the wiki portion, the group must generate documentation describing their system using the wiki functionality provided by GitLab. The wiki must contain at least one page that provides a high-level description of the entire system and must contain at least one page for each of the major functionality components (i.e. if there are six major functionality components, there should be at least six pages).

For the high-level description page, the team must provide a brief description of each of the major functionality components, how they work together, and some of the major data structures across the entire project. This information should not be copied and pasted from the project instructions. The page must also contain a diagram of the entire system, based on the one created during recitations. The diagram must be created digitally (i.e. using PowerPoint, Photoshop, Paint.net, UMLet, etc.), must be easy to read and understand, and cannot a photographed or scanned image.

For each major functionality component page, the student accountable for that component must provide a detailed description of their component. This description should have three labeled sections: a brief description of the purpose of the component, a description of how data was stored and maintained for this component, and a description of the functionality for the component. They might also consider including diagrams to more easily visualize how all of the pieces fit together.

For the data storage and maintenance section, there should be an explanation of how data was stored and maintained in their component. What, if any, objects or structs were created to store data? How were they organized and managed? What types of formal data structures did were made use of (trees, graphs, arrays, hashes, etc)?

For the functionality component, there should be an explanation of the major functions in the component. How is data moved and transformed? How is it read in? How is it output? What are the various major functions constructed and how do they work?

Descriptions and explanations should be primarily in prose and paragraph format, not bulleted lists. Code snippets are also acceptable, but must be used as an enhancement to the explanation of functionality not as a substitution for it. Your grade for the wiki will partly be based on apparent effort, so please be thorough in your descriptions. Additionally, because this is a wiki, the high-level description page must have links to all of the major functionality component pages.

Implementation

Your program must provide the following functionality and adhere to the following constraints:

- Your int main() should be in its own .c/.cpp file
- All .c/.cpp files, except your main.c/main.cpp, must have associated header files. You may not #include a .c/.cpp file, only header files
- Allow the user to input the file containing the simulation configuration
    - Do **NOT** hardcode the filename into your program
    - The first line will provide the name of the file containing the region population (Do **NOT** prompt the user for this filename)
    - The second line will provide the name of the file containing the region layout (Do **NOT** prompt the user for this filename)
    - The third line will be empty
    - The fourth line will provide information regarding which area in the region starts with the infection
    - The fifth line will provide information regarding the infectious period of the disease
    - The sixth line will provide information regarding the contact rate of the disease
    - The seventh line will provide information regarding the number of available vaccines
- Your system should perform the following operations:
    - Read in and store the simulation configuration information
    - Read in and store the initial region population
        - Each line contains the ID of the area and its population
        - There can be any number of areas in a region, and they can have any population
    - Read in and store the region layout
        - The region layout stores the region as an adjacency matrix with a header on the top row and left column
        - The region can be considered to be a contiguous, planar graph where areas are nodes and an edge between two nodes indicates those two areas share a border
        - A 1 in a cell indicates two areas are adjacent
        - A 0 in a cell indicates two areas are not adjacent
        - The adjacency matrix will always be square
    - Output the ID and population of each area
    - Output an adjacency list representing the region
    - Setup the following experiments
        - Distribute vaccines amongst the areas according to closeness centrality, smaller closeness values have priority over larger closeness values. In the case of a tie between closeness centrality

values, smaller label values take priority. Closeness is defined as the average shortest distance between an area and every other area. You should attempt to distribute the maximum number (population size) of vaccines to an area before beginning distributing vaccines to the next area

    - Distribute vaccines amongst the areas according to degree centrality, larger degree values have priority over smaller degree values. In the case of a tie between degree centrality values, smaller label values take priority. Degree for an area is the number of adjacent areas. You should attempt to distribute the maximum number (population size) of vaccines to an area before beginning distributing vaccines to the next area
    - Distribute vaccines amongst the areas randomly. The areas labels have already been randomized so allocate vaccines starting at the smallest label and working to the largest label. You should attempt to distribute the maximum number (population size) of vaccines to an area before beginning distributing vaccines to the next area
    - Distribute vaccines amongst the areas equally. Each area should receive an equal share of the vaccine. An area cannot receive more vaccines than it has people, and any remaining vaccines from its allocation should be distributed round-robin style to the areas starting with the lowest numbered area and working to the highest numbered area.
- Execute each experiment using an agent-based SIRV model
    - In SIRV models, we say that agents have one of four health states: Susceptible, Infectious, Recovered, Vaccinated. Susceptible agents do not have the disease but can contract it. Infectious agents have the disease and are spreading it for the length of their infectious period. Recovered agents are no longer infectious and have immunity to the disease. Vaccinated agents cannot contract the disease due to being vaccinated
    - In agent based models, each individual is represented as a separate entity, not just a population total. This means they will each agent will have their own health states and their own infectious period
    - For each experiment, one agent in the initially infected area will be infectious on Day 0. This does count as a day for the infectious period of that agent
    - Before an experiment is executed, the name of that distribution method should be output
    - For each experiment, your simulator must output for each day (including day 0)
        1. The current simulated day number
        2. The id, total population, number of susceptible, number of infectious, number of recovered, and number of vaccinated agents for each area
    - Each day, every infectious agent in an area infects contact rate number of available susceptible agents. These agents do not become infectious until the next day
    - At the end of the day, any infectious agents that have spent a number of days equal to the infectious period should become recovered for the next day. This means that every infectious agent should infect susceptibles for the full number of infectious period days
    - Recovered and vaccinated agents do not change states
    - If an area ever has more than 50% of its population infectious in a single day, then it infects a single person in each of its adjacent areas that have no infectious agents. These agents do not become infectious until the next day
    - The experiment stops when there are no more infectious agents in the region

    - Collect and output data regarding the experiment
        - For each experiment, you should determine what the peak number of infectious agents was, when the first day that peak occurred, what day the outbreak ended on (i.e. the first day with no infectious agents in the region), and how many agents in total were infected
        - This data should be output as a summary after all experiments have been executed
- See the example output files for formatting of each of the outputs
- Major functionality components must be constructed in some function, or across some functions, that are declared and defined outside of your main.c/main.cpp . Remember, function declarations must be stored in a header file, while definitions must be stored in a .c/.cpp file. You may have additional functions that support your major functionality component function.
- Your code must be well commented.
- Each group member should be performing regular commits to the GitLab repository on the Apollo server with meaningful commit messages. “Fixed bug” or “New code” are not meaningful, so try to be more specific about what was fixed or what was added.
- Please do not commit the example input and output files to the remote server as that may cause a space issue on the server.
- You must provide a .txt format README file which includes:
    - The names of all group members
    - Instructions on how to compile your program
    - Instructions on how to run your program
    - An indication on whether you implemented the bonus or not. By default, the TA’s will assume you did not attempt the bonus unless you indicate otherwise.
- Additionally, you may write a makefile, but please specify if it needs any additional flags to function properly

Each student must be accountable for one or more major functionality components, and may not swap after they sign up for a component barring an exceptional circumstance. Failure to be accountable for any major functionality component will result in a 0 for the coding portions of the project (milestone submission and/or final submission). Keep in mind that some components build on others, so be careful about who takes ownership of which pieces and manage your time to avoid a crunch near the due date. Also, the group should strive to balance the work across all members. The major functionality components are:

1. Reading in the configuration file, region file, and population file, storing and organizing the data, and outputting the adjacency list of the graph representing the region along with the population of each area
2. Calculating the closeness centrality and determining vaccine distribution for each area
3. Calculating the degree centrality and determining vaccine distribution for each area
4. Determining the vaccine allocation for each area for random distribution
5. Determining the vaccine allocation for each area for equal distribution
6. Setting up the simulation for each distribution strategy
7. Executing an agent-based SIRV simulation for each distribution strategy
8. Collecting, analyzing, and outputting desired results

Milestone Project Submission

Your program must provide all requested functionality for major functionality component 1 (reading in the configuration file, region file and population file and storing and organizing the data), as well as outputting the adjacency list of the graph representing the region along with the population of each area. At least one group member must submit a .zip file containing the following:

1. All files necessary to compile and run your program (Please do not include any files not necessary to run the program on the CSE machines)
2. A .txt format README file explaining how to compile and run your program

Final Project Submission

Your program must provide all requested functionality. At least one group member must submit a .zip file containing the following:

1. All files necessary to compile and run your program (Please do not include any files not necessary to run the program on the CSE machines)
2. A .txt format README file explaining how to compile and run your program

Rubric

The entire assignment is worth 300 points, and each student will receive a single grade with respect to both the entire project and to the portions they were individually responsible for. The breakdown of those points is as follows.

- 20 points: Project Expectations Document
- 30 points: Project Design
- 15 points: Project Check-ins
- 100 points: Project Milestone Submission
- 100 points: Project Final Submission
    - 80 points: Implemented functionality
    - 10 points: Proper Commenting
    - 10 points: Group Evaluation
- 10 points: GitLab Commits
- 25 points: Project Wiki

If your code fails to compile on the CSE machines you will not receive credit for the code portion of the assignment (milestone submission and/or final submission). I recommend not making changes to your code without checking for compilation before you submit.