CS 201, Fall 2024
Homework Assignment 4

Due: 23:59, December 24, 2024

1 Introduction

Welcome to the City of Contagion!

Hold tight to your face masks — it's time to simulate the spread of a pandemic in a city, and only you will uncover its fate!
Imagine a city represented as an $m \times n$ grid, where each cell tells a story:

- **0** : An empty block, perhaps a park, or maybe just an abandoned building.
- **1** : A block with healthy individuals, living their best lives (at least for now).
- **2** : An infected block with infected individuals, ready to spread the infection.

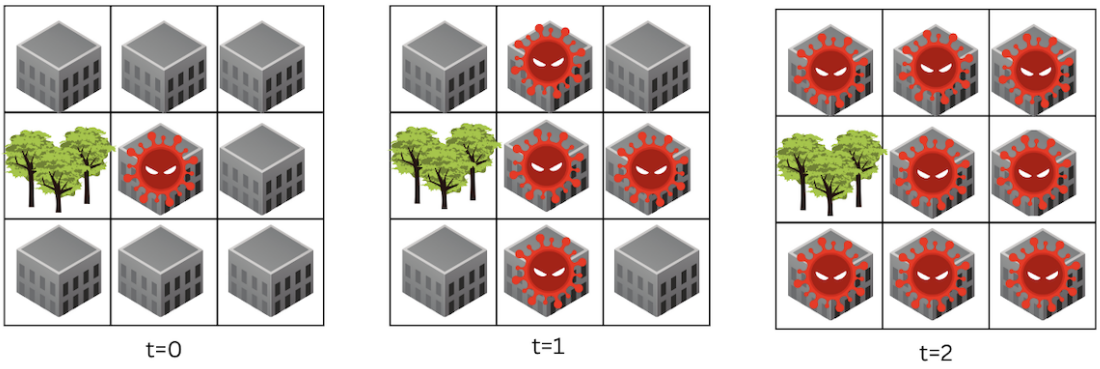

Figure 1: Spread of the pandemic at each time step

Every day, the infection spreads from each infected block to any adjacent **healthy block** only, horizontally or vertically. It cannot spread diagonally and it cannot spread to empty blocks. Your mission, should you choose to accept it:

> Calculate the minimum time (number of days) it will take for the infection to reach every healthy block in the city. If the infection cannot reach all healthy blocks (because of empty blocks), state that as described in output section. Sorry, some people are just out of reach! Are you ready to discover the fate of the city? The clock is ticking!

1.1 Input Specifications

We will provide you with a single file, `cityGridFile`, which contains the following information:

- The first line will provide the dimensions of the city grid in the format: `m n`, where `m` is the number of rows and `n` is the number of columns.
- The subsequent lines will represent the grid information, where each grid block corresponds to one of three possible states:

  0 : An empty block (no building or person),

  1 : A healthy block,

  2 : An infected block.

For example, the `cityGridFile` file for Figure 1 would look like this:

```
3 3
111
021
111
```

1.2 Class Implementation

Your solution must be implemented in a class called **PandemicSimulator**. Below is the required public part of the `PandemicSimulator` class. The interface for the class must be written in a file called **PandemicSimulator.h** and its implementation must be written in a file called **PandemicSimulator.cpp**. You can define additional public and private member functions and data members in this class. You can also define additional classes in your solution.

```
class PandemicSimulator {

public:
    PandemicSimulator(const string cityGridFile);
    ~PandemicSimulator();

    void displayCityState(const int time);
    void simulateBlock(const int row, const int col);
    void simulatePandemic();

};
```

The member functions are defined as follows:

- `PandemicSimulator` Constructor. Reads the city information from the `cityGridFile` and stores the city grid. There can be multiple infected blocks (cells) in the grid. Infection should start spreading simultaneously from those blocks. You can assume that there will be at least one infected block in any of the inputs. You can also assume that the input file is correctly formatted.

- `displayCityState` Displays the states of all cells in the city grid at a given time. When given time is 0 it should print the initial state of the city.

- `simulateBlock` Displays the minimum number of days required for the given block to be infected. If the given cell corresponds to an empty block, it should display -1. If the given cell is already infected in the input, it should display `0`. Note that the block (cell) coordinates are defined just like the row and column indices of a 2D array.

- `simulatePandemic` This function simulates the spread of the pandemic across the entire city grid, starting from all initially infected cells simultanously. The infection spreads across adjacent cells horizontally and vertically at each time step, and the function calculates and outputs the minimum time required for the infection to reach all healthy blocks. If it is impossible for the infection to reach all healthy blocks, the function should output `Pandemic cannot spread to all blocks.`

1.3 Example Test Case and Output

This section walks through an example testcase and the corresponding output. We will use a similar program to test your solution, so make sure that the name of the class is **PandemicSimulator**, its interface is in the file called **PandemicSimulator.h**, and the required functions are defined as shown above.

Input File: `cityGridFile.txt`

This is the input file used for testing the PandemicSimulator:

```
6 6
110000
121100
100111
110010
011120
000111
```

Above input represents a 6×6 city grid where there are 2 infected blocks at indices (1,1) and (4,4) initially. In the given scenario, the infection begins at the cells marked as '2' in the grid, and spreads across adjacent cells horizontally and vertically. It does not spread diagonally. You can assume that all inputs will be valid.

Example Test Program

```cpp
#include "PandemicSimulator.h"

int main() {

    PandemicSimulator ps("cityGridFile.txt");

    ps.displayCityState(0);
    cout << endl;

    ps.displayCityState(1);
    cout << endl;

    ps.displayCityState(2);
    cout << endl;

    ps.simulateBlock(0, 0);
    ps.simulateBlock(4, 4);
    ps.simulateBlock(0, 4);
    ps.simulateBlock(1, 2);
    ps.simulateBlock(3, 1);
    cout << endl;

    ps.displayCityState(3);
    cout << endl;

    ps.displayCityState(4);
    cout << endl;

    ps.displayCityState(5);
    cout << endl;

    ps.simulatePandemic();

    return 0;
}
```

Expected Output

```
City state at day 0:
110000
121100
100111
110010
011120
000111

```

```
City state at day 1:
120000
222100
100111
110020
011220
000121

City state at day 2:
220000
222200
200121
110020
012220
000222

Time for block (0, 0) to be infected: 2 days.
Time for block (4, 4) to be infected: 0 days.
Time for block (0, 4) to be infected: -1 days.
Time for block (1, 2) to be infected: 1 days.
Time for block (3, 1) to be infected: 4 days.

City state at day 3:
220000
222200
200222
210020
022220
000222

City state at day 4:
220000
222200
200222
220020
022220
000222

City state at day 5:
220000
222200
200222
220020
022220
000222

Minimum time for pandemic to spread to all blocks: 4 days.
```

If there are still healthy blocks left after running the simulation, the **simulatePandemic**

function should output the following:

```
Pandemic cannot spread to all blocks.
```

This indicates that some healthy blocks are unreachable, and the simulation could not infect all of them.

2 Specifications

1. You ARE NOT ALLOWED to modify the given parts of the header file. You MUST use the nonrecursive solution using a queue to implement the Breadth-First Search (BFS) algorithm for simulating the infection spread in the city grid. You will get no points if you use any other algorithm for the solution of the infection spread problem. You can use other data structures to help with your implementation, but the main simulation algorithm must be implemented using a queue.

2. You MUST implement the queue and any additional container (e.g., list) by yourself. You ARE NOT ALLOWED to use the data structures and related functions in the C++ standard template library (STL) or any external library.

3. Moreover, you ARE NOT ALLOWED to use any global variables or any global functions.

4. Output message for each operation MUST match the format shown in the output of the example code.

5. Your code MUST NOT have any memory leaks. You will lose points if you have memory leaks in your program even though the outputs of the operations are correct. To detect memory leaks, you may want to use Valgrind which is available at `http://valgrind.org`.

3 Submisson

1. In this assignment, you must have separate interface and implementation files (i.e., separate `.h` and `.cpp` files) for your class. Your class name MUST BE `PandemicSimulator` and your file names MUST BE `PandemicSimulator.h` and `PandemicSimulator.cpp`. Note that you may write additional class(es) in your solution.

2. The code (`main` function) given above is just an example. We will test your implementation using different scenarios, which will contain different function calls. Thus, do not test your implementation only by using this example code. We recommend you to write your own driver files to make extra tests. However, you MUST NOT submit these test codes (we will use our own test code). In other words, do not submit a file that contains a function called `main`.

3. You should put all of your `.h` and `.cpp` files into a folder and zip the folder (in this zip file, there should not be any file containing a `main` function). The name of this zip file should conform to the following name convention: secX-Firstname-Lastname-StudentID.zip where X is your section number. The submissions that do not obey these rules will not be graded. Please do not use Turkish letters in your file and folder names.

4. Make sure that each file that you submit (each and every file in the archive) contains your name, section, and student number at the top as comments.

5. You are free to write your programs in any environment (you may use Linux, Windows, MacOS, etc.). On the other hand, we will test your programs on "dijkstra.ug.bcc.bilkent.edu.tr" and we will expect your programs to compile and run on the dijkstra machine. Your code will be tested by using an automated test suite that includes multiple test cases where each case corresponds to a specific number of points in the overall grade. We will provide you with example test cases by email. Thus, we strongly recommend you to make sure that your program successfully compiles and correctly works on dijkstra.ug.bcc.bilkent.edu.tr before submitting your assignment. If your current code does not fully compile on dijkstra before submission, you can try to comment out the faulty parts so that the remaining code can be compiled and tested during evaluation.

6. This assignment is due by 23:59 on Tuesday, December 24, 2024. You should upload your work to Moodle before the deadline. No hardcopy submission is needed. Late submissions will not be accepted (if you can upload to Moodle, then you are fine). There will be no extension to this deadline.

7. We use an automated tool as well as manual inspection to check your submissions against plagiarism. For questions regarding academic integrity and use of external tools (including generative AI tools), please refer to the course home page and the Honor Code for Introductory Programming Courses (CS 101/102/201/202) at `https://docs.google.com/document/d/1v_3ltpV_lClLsROXrMbojyuv4KrFQAm1uoz3SdC-7es/edit?usp=sharing`.

8. This homework will be graded by your TA **Sude Önder** (sude.onder@bilkent.edu.tr). Thus, you may ask your homework related questions directly to her. There will also be a forum on Moodle for questions.