CS 202, Spring 2025

Homework Assignment 4

Due: 23:59, May 16, 2025

Homework Submission Instructions

Before you start your homework, please read the following instructions carefully:

FAILURE TO FULFILL ANY OF THE FOLLOWING REQUIREMENTS WILL RESULT IN A GRADE SCORE OF 0 (zero) WITHOUT ANY CHANCE OF REDEMPTION.

- See the course page for any late submission policies and Honor Code for Assignments.
- Upload your solutions in a single ZIP archive using the Moodle submission form. Name the file as `studentID_name_surname_hw4.zip`.
- Your ZIP archive should contain only the following files:
  - `CyberAttackContainment.cpp`, `CyberAttackContainment.h`
  - You can have auxiliary `.cpp` and `.h` files.
  - Your files should not contain any implementation of `main` function.
- Do not forget to put your name, student ID, and section number in all of these files. Add the following header to the beginning of each file:

```
/*
 * Author : Name & Surname
 * ID: 12345678
 * Section : 2
 * Homework : 4
 */
```

- Your code must be compilable.
- Your code must be complete.
- You **ARE NOT ALLOWED** to use any data structure or algorithm-related function from the C++ Standard Template Library (STL) or any external libraries.
- Your code must contain the proper comments needed to understand its function.
- Your code must be run on the server `dijkstra.cs.bilkent.edu.tr`.
- Your code is expected to successfully pass the provided test cases. If none of the test cases are passed, you will get zero.
- You MUST use a nonrecursive solution using queue to implement the breadth-first search algorithm for computing the spread of malware in the network.
- For any questions related to the homework, contact your TA: `saeed.karimi@bilkent.edu.tr`.

DO NOT START YOUR HOMEWORK BEFORE READING THIS SECTION CAREFULLY! THE INSTRUCTIONS MAY DIFFER FROM HOMEWORK TO HOMEWORK!

1 Cyberattack Containment in a Distributed Network

In the wake of a targeted cyberattack, a large-scale computer network has been partially compromised by a rapidly spreading malware. The network is modeled as an undirected graph $G = (V, E)$, where:

- Each node represents a computing device.
- Each edge represents a secure communication link between devices, annotated with a latency value (edge weight), indicating the time it takes for security updates to propagate.

Some nodes in the network were initially compromised by the attacker — these are labeled as Malware Injection Points. Simultaneously, a defense protocol has been activated from designated nodes called Security Patch Dispatchers, responsible for distributing patches that can immunize nodes from infection. The malware spreads aggressively but uniformly, moving to all neighboring nodes in unit time per hop (i.e., one hop per minute). In contrast, security patches propagate along latency-optimized paths (computed using the provided edge weights as the link latency time).

2 Your Task: Predict node status and safe zones

After the malware and patch propagation complete:

- A node is considered **Infected** if the malware reaches it strictly before the patch.
- A node is considered **Secured** if the patch reaches it before or at the same time as the malware.

To prevent further contamination, all communication links between infected and secured nodes are removed — effectively partitioning the network into isolated safe and infected zones.

Your job is to:

- Label each node as infected or secured.
- Remove all edges connecting an infected node to a Secured node.
- In the resulting graph, identify the safe and infected zones.

Why This Matters: In real-world cyber-defense systems, early detection and response time are critical. Understanding whether a node can be protected — and isolating infected regions of a network — is key to limiting malware spread. The ability to dynamically partition a network into safe and unsafe regions ensures that safe zones remain operational while infected zones are considered for investigation.

3 An illustrated example

In this section, we provide an example of the network graph (Figure 1) and the corresponding output graph (Figure 2) that you are expected to produce. Additionally, we describe the input file format, the class prototype, a sample test code, and the expected output of the test code.

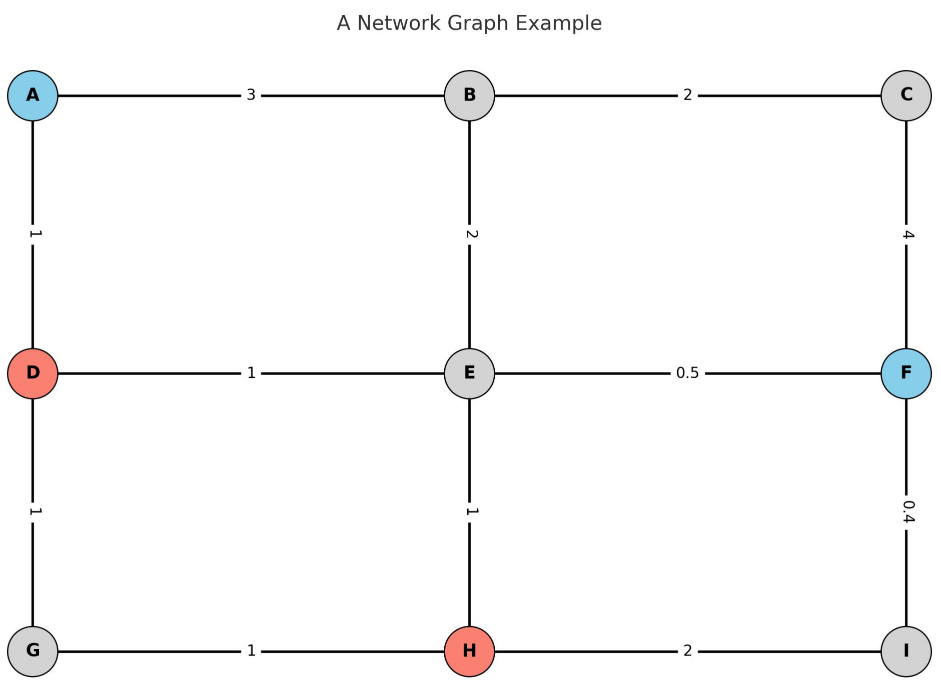

Figure 1: An example of the input graph is shown. Gray nodes represent normal nodes, blue nodes represent dispatcher nodes, and orange nodes represent malware nodes.

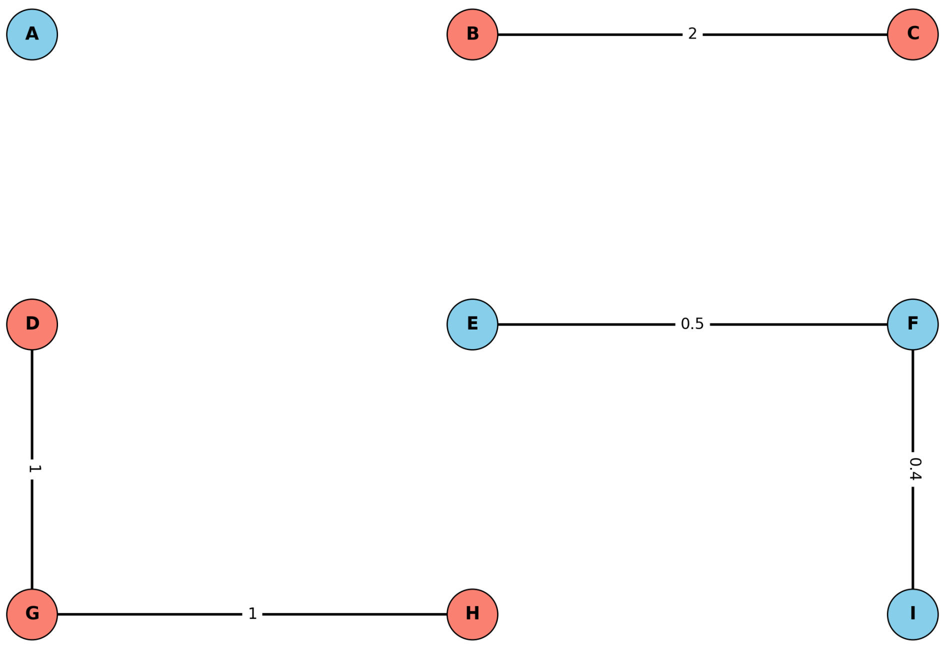

Figure 2: An example of the output graph is shown. Each connected component represents either a secured or an infected zone. Blue nodes correspond to secured zones, while orange nodes indicate infected zones.

3.1 Input file

Below is an example of an input file named `network.txt`. The first line specifies the total number of nodes. Each subsequent line lists a node's name along with its type. Finally, the edges and their associated weights (representing the time latency for patch propagation) are provided, one per line.

```
9
A dispatcher
B normal
C normal
D malware
E normal
F dispatcher
G normal
H malware
I normal
A B 3
B C 2
A D 1
B E 2
C F 4
D E 1
E F 0.5
D G 1
E H 1
F I 0.4
G H 1
H I 2
```

3.2 Class prototype

```
class CyberAttackContainment {

   public:
      CyberAttackContainment(const string inputFile);
      ~CyberAttackContainment();

      // Prints the infection status (Secured or Infected) for each node
      void nodeStatuses();

      // Creates and prints connected components, labeling them as Secured or
          Infected Zones
      void computeConnectedZones();

};
```

The member functions are defined as follows:

CyberAttackContainment: The constructor reads the network information from the file `network.txt` provided as a parameter. You may assume that the file contains at least one node (i.e., it is not empty) and that its contents are formatted correctly.

nodeStatuses: Determines the status of each node after the malware and patch propagation. Then, it prints each node along with its corresponding status on a separate line. The nodes are listed in ascending alphabetical order based on their names.

computeConnectedZones: Computes and prints the secured and infected zones. Each zone is displayed on a single line, with its nodes listed in ascending alphabetical order, followed by the zone's status at the end. Zones themselves are printed in ascending order based on their representatives, where the representative is defined as the first node in the zone's alphabetically sorted list.

3.3 Example test code

```
#include "CyberAttackContainment.h"

int main() {

    CyberAttackContainment cac("network.txt");
    cout<<endl;
    cac.nodeStatuses();
    cout<<endl;
    cac.computeConnectedZones();

    return 0;
}
```

3.4 Output of the example test code

```
9 nodes and 12 connections are loaded.

Node states are:
A secured
B infected
C infected
D infected
E secured
F secured
G infected
H infected
I secured

Connencted zones are:
A secure
B C infected
D G H infected
E F I secured
```