[CENG 315 ALL Sections] Algorithms

THE7

Available from: Saturday, December 23, 2023, 12:00 PM
Due date: Sunday, December 24, 2023, 11:59 PM
Requested files: the7.cpp, test.cpp (Download)
Maximum number of files: 3
Type of work: Individual work

In your network security term project, you are tasked with planning a network attack. You propose a method to your friends that aims to maximize the speed of infecting the whole network. Given a network, you will first calculate the ***infection_score*** for each node, which represents ***how fast the whole network will be infected if you only infect the selected node***.

The network is represented as a **directed, weighted graph**, where the **weights of each edge represent how long it takes the network to deliver a package** between the two nodes, i.e. the vertices of that edge. For node count *N*, and the maximum shortest path distance in the graph between any pair *(i,j)* as *MaxDist*, ***infection_score*** "*IS*" is defined as follows:

Infection score (IS) for node i:

$$IS(i) = \frac{1}{AIS(i)}$$

Average infection speed (AIS) for node i:

$$AIS(i) = \frac{\sum_{j=0, j\neq i}^{N} SP(i,j)}{N-1}$$

Definition of SP(i,j):

$$SP(i,j) = \begin{cases} MaxDist + 1 & \text{if there is no path between } (i,j) \\ \text{shortest distance between } (i,j) & \text{otherwise} \end{cases}$$

Problem

In this exam, you are asked to calculate the ***infection_scores*** given the ***network*** as a **directed, weighted graph** by completing the ***get_infection_scores()*** function defined below.

```
void get_infection_scores(const std::vector< std::vector<std::pair<int, int>>>
&network, std::vector<double> infection_scores));
```

- ***network*:** Graph adjacency list
- ***infection_scores*:** Calculated infection scores (*IS*) of each node, ordered by node ID.

Constraints and Hints:

- Carefully examine the definition of *SP(i,j)*. SP returns the shortest **directed** path distance between two nodes *(i,j)*. If there is no directed path between *(i,j)*, instead, it returns the maximum shortest distance in the network between any two pairs + 1. This way, nodes are penalized for not having a connection to other nodes.
- Be careful when calculating the average infection speed *AIS*. You should not include a self-path for a node in your calculation, and hence, you should divide the sum of *SP(i,j)* by **N**-1.
- Limits for **N** where 1 < **N** <= 500.
- The weight **w** of each edge is between 1 <= **w** <= 50

Evaluation:

- After your exam, black-box evaluation will be carried out. You will get full points if you return the correct infection scores for each node. The grade you see in the VPL contains 50% of your final grade. We will evaluate your grades with different inputs after the end of the exam.
- Note: If your implementation does not return before the given time limit per case, VPL will show "incorrect" as your output. If you believe your implementation is correct value-wise, please check if it runs below the time limit.

Example IO:

1)

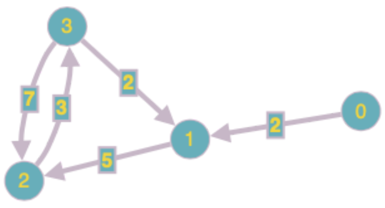
Network Structure:

0: { (1, 2) }

1: { (2, 5) }

2: { (3, 3) }

3: { (1, 2) (2, 7) }



Infection scores: 0.157895 0.125 0.157895 0.15

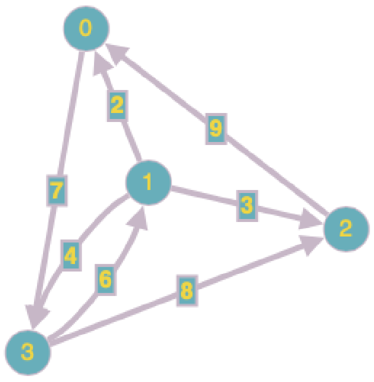
2)

0: { (3, 7) }

1: { (3, 4) (2, 3) (0, 2) }

2: { (0, 9) }

3: { (2, 8) (1, 6) }



Infection scores: 0.0857143 0.333333
0.0638298 0.136364

Specifications:

- There is 1 task to be solved in **36 hours** in this take-home exam.
- You will implement your solutions in ***the7.cpp*** file.
- You are free to add other functions to *the7.cpp*
- **Do not change** the first line of the7.cpp, which is #include "the7.h"
- *<vector>, <queue>, <stack>, <climits>, <algorithm>, <utility> and <memory>* are included in "the7.h" for your convenience, you can use them freely.
- **Do not change** the arguments and the return value of the function ***get_infection_scores()*** in the file the7.cpp
- **Do not include** any other library or write include anywhere in your the7.cpp file (not even in comments).
- You are given ***test.cpp*** file to test your work on **ODTUClass** or your **locale**. You can, and you are, encouraged to modify this file to add different test cases.
- If you want to test your work and see your outputs you can compile your work on your locale as:

```
>g++ test.cpp the7.cpp -Wall -std=c++11 -o test

> ./test
```

- You can test your the7.cpp on the virtual lab environment. If you click **run**, your function will be compiled and **executed with test.cpp**. If you click **evaluate**, you will get **feedback** for your current work and your work will be **temporarily graded** for a limited number of inputs.
- The grade you see in lab is not your final grade, **your code will be reevaluated with different inputs** after the exam.
- You can download the sample IO from here.

The system has the following limits:

- a maximum execution time of 3 second per test case

- a 1 GB maximum memory limit,
- an execution file size of 4M.
- Solutions with longer running times will not be graded.
- If you are sure that your solution works in the expected complexity, but your evaluation fails due to limits in the lab environment, the constant factors may be the problem.

Requested files

the7.cpp

```cpp
#include "the7.h"

// do not add extra libraries here

void get_infection_scores(const std::vector<std::vector<std::pair<int, int>>>& network,
                          std::vector<float>& infection_scores){

                          }

```

test.cpp

```cpp
#include <iostream>
#include <fstream>
#include "the7.h"


void print_network(std::vector<std::vector<std::pair<int,int>>>& network) {
    int node_number = (int) network.size();
    if (node_number == 0) {
        std::cout << "There is no node in the network" << std::endl;
        return;
    }

    for (int idx=0; idx < node_number; idx++) {
        std::cout << idx << ":\t{";
        for (const auto& edge : network[idx]) {
            std::cout << " (" << edge.first << ", " << edge.second << ") ";
        }
        std::cout << "} " << std::endl;
    }
}

void read_from_file(std::vector<std::vector<std::pair<int, int>>>& network){
    int node_number, edge_number;
    char addr[]= "inp00.txt"; // 01-10 are available
    std::ifstream infile (addr);
    if (!infile.is_open()){
        std::cout << "File \'"<< addr
                  << "\' can not be opened. Make sure that this file exists." << std::endl;
        return;
    }
    infile >> node_number >> edge_number;
    network.resize(node_number);
    for(int idy=0; idy < edge_number; idy++) {
        int source, dest, weight;
        infile >> source >> dest >> weight;
        network[source].push_back(std::make_pair(dest, weight));
    }
    infile.close();
}

int main(){
    std::vector<std::vector<std::pair<int, int>>> network;
    std::vector<float> infection_scores;
    read_from_file(network);
    print_network(network);
    get_infection_scores(network, infection_scores);
    std::cout << "Infection scores: ";
    for(const auto& score : infection_scores) std::cout << score << " ";
    std::cout << std::endl << "-------------------------------------------" << std::endl;
    return 0;
}
```

VPL