[CENG 315 ALL Sections] Algorithms

Navigation

- Dashboard
  - Site home
  - Site pages
  - My courses
    - 571 - Computer Engineering
      - CENG 223 All Sections
      - CENG 223 Section 2
      - CENG 315 ALL Sections
        - Participants
        - Competencies
        - Grades
        - General
        - October 2 - October 8
        - October 9 - October 15
        - October 15 - October 22
        - October 23 - October 29
        - October 30 - November 5
        - November 6 - November 12
        - November 13 - November 19
        - November 20 - November 26
        - November 27 - December 3
        - December 3 - December 10
        - December 11 - December 17
          - THE7
            - Description
            - Submission
            - Edit
            - Submission view
          - THE7 Discussion
        - December 18 - December 24
        - December 25 - December 31
        - January 1 - January 7
        - January 8 - January 14
      - CENG 315 Section 1

Description | Submission | Edit | Submission view

Grade

Reviewed on Sunday, December 24, 2023, 12:16 PM by Automatic grade
Grade: 100.00 / 100.00

Assessment report [-]
[-]**Output of make**
g++ -c -o sol7.o sol7.cpp
[-]**For input 01:**
Infection scores: correct
[-]**For input 02:**
Infection scores: correct
[-]**For input 03:**
Infection scores: correct
[-]**For input 04:**
Infection scores: correct
[-]**For input 05:**
Infection scores: correct
[-]**For input 06:**
Infection scores: correct
[-]**For input 07:**
Infection scores: correct
[-]**For input 08:**
Infection scores: correct
[-]**For input 09:**
Infection scores: correct
[-]**For input 10:**
Infection scores: correct

Submitted on Sunday, December 24, 2023, 12:06 PM (Download)

the7.cpp

```cpp
#include "the7.h"

// do not add extra libraries here

const int INF = 1e9; // Set a large value to represent infinity
void get_infection_scores(const std::vector<std::vector<std::pair<int, int>>>& network,
                          std::vector<float>& infection_scores)
{
    int n = network.size();
    std::vector<std::vector<int>> shortest_paths;
    shortest_paths.assign(n, std::vector<int>(n, INF));

    // Initialize shortest_paths with network
    for(int i = 0; i < n; ++i) {
        shortest_paths[i][i] = 0;
        for(auto &edge : network[i]) {
            int j = edge.first;
            int w = edge.second;
            shortest_paths[i][j] = w;
        }
    }

    int max_shortest_distance=0;
    // Floyd-Warshall algorithm
    for(int k = 0; k < n; ++k) {


        for(int i = 0; i < n; ++i)
        {
            double temp =0;
            for(int j = 0; j < n; ++j)
            {
                if((shortest_paths[i][j] > (shortest_paths[i][k] + shortest_paths[k][j])))
                {
                    shortest_paths[i][j] =  shortest_paths[i][k] + shortest_paths[k][j];
                }
                if( shortest_paths[i][j] != INF && max_shortest_distance<shortest_paths[i][j])
                {
                   max_shortest_distance = shortest_paths[i][j] ;
                }
            }
        }


    }

    for(int i = 0; i < n;i++)
    {
        double temp =0;
        for(int j = 0; j < n;j++)
        {
            if(shortest_paths[i][j] == INF)
            {
                temp += max_shortest_distance+1;
            }
            else if(i != j)
            {
                temp +=shortest_paths[i][j];
            }
```

```
        }
        infection_scores.push_back(1/(temp/(n-1)));
    }
}

```

test.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[]= "inp10.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