CENG 301, Spring 2024
Homework Assignment 4

Due: 23:59, June 10, 2024

In this homework, you will implement a Graph class, note that this graph will be undirected. This Graph class will have the following functionalities. The details of these functionalities are given below:

1. Representing a graph with adjacency matrix
2. Finding shortest path with Dijkstra
3. Finding the connected components in the graph, finding centroid nodes for each connected component, and listing all centroid nodes
4. Finding the connected component of a given node belongs to
5. Determining whether a node is centroid or not, listing nodes in a connected component

Representing a graph with adjacency matrix: Given a graph represented as a list of edges, number of nodes and number of edges, store the graph in adjacency matrix (Constructor function)

Finding shortest path with Dijkstra: Perform Dijkstra algorithm with given source node. Find its distance to all other nodes. Return these distances as an array. The length of the array will be equal to number of nodes in the graph. If it is not possible to reach a node, you may put -1 to corresponding index in returned distance array. (function name: *dijkstra*)

Finding the connected components in the graph, finding centroid nodes for each connected component, and listing all centroid nodes: Each connected component has a "centroid node". Centroid node of a connected component is defined as follows: the node that is in the connected component and minimizes the sum of distances to all other nodes in the connected component. Your code should be able to return centroids as an array. The 0th element of returned array is the number of centroids (since each connected component has only one centroid, number of centroids is equal to number of connected components), remaining elements are the centroid nodes sorted in ascending order according to their node IDs. For example 1st elemnt of the returned array will be the centroid node with the smallest node ID. Length of the returned array should be one more than the 0th element of returned array since 0th element is the number of centroids. (function name: *get_centroids*)

Finding the connected component of a given node belongs to: Given a node, your code should be able find which connected component does this node belong to, and your code should return the centroid of the corresponding connected component. (function name: *get_centroid_of_connected_component_containing*)

Determining whether a node is centroid or not, listing nodes in a connected component: Given a node, your code should determine whether it is a centroid or not. If it is, return an array describing the connected component. 0th element of this array will be equal to number of nodes in this connected component, remaining elements are the nodes in the connected component sorted in ascending order according to their node IDs. If the node is not centroid, return an array of length one, where 0th element is 0. (function name: *nodes_in_connected_component_with_centroid*)
Below is the headers for `Edge` and `Graph`, classes that you must write in this assignment. You should implement the `Graph` class in a separate .cpp file.

```
// Edge.h
// DO NOT MODIFY THIS FILE
// EVEN THOUGH YOU TRY TO MODIFY, YOUR MODIFICATIONS WILL NOT BE CONSIDERED IN GRADING
```

```
#ifndef EDGE_H
#define EDGE_H
class Edge
{
public:
    int node1;
    int node2;
    int length;
};
#endif
```

```
// Graph.h
/*
    DON'T CHANGE THE EXISTING FUNCTION SIGNATURES AND VARIABLE NAMES
    IF YOU NEED, YOU CAN ADD NEW FUNCTIONS AND VARIABLES
*/
#ifndef GRAPH_H
#define GRAPH_H
#include "Edge.h"

class Graph
{
private:
    /*
    Since arrays are indeed pointers, you may use a double pointer as a matrix.
    You can allocate it using new or malloc (multiple new or malloc operations may be
        needed.).
    */
    int** adjMatrix;
    int num_nodes;
    int num_edges;
    /* ADD MORE PRIVATE VARIABLES/FUNCTIONS IF NEEDED */

public:
    Graph (Edge* edges_array, int num_nodes, int num_edges);
    ~Graph();
    /*
    dijkstra function returns a dynamically allocated array of integers, corresponding
         to distances from source node to nodes.
    For example 0th element of the returned array is the distance from source node to
        node 0.
    If it is not possible to reach a node, you may add -1 to the array as an
        exceptional value.
    If the length of the returned array is not equal to number of nodes, you may get a
         segmentation fault.
    */
    int* dijkstra(int source);

    /*
    get_centroids function returns a dynamically allocated array of integers,
    0th element of the returned array will be number of connected components
    The other elements are centroid nodes sorted in ascending order.
    For example the 1st element of the returned array is the centroid node with the
        smallest value.
```

```
    Let's name the returned array as returned_array,
    If the length of the returned array is not equal to returned_array[0]+1, you may
        get a segmentation fault
    */
    int* get_centroids();


    int get_centroid_of_connected_component_containing(int node);

    /*
    nodes_in_connected_component_with_centroid function returns a dynamically
        allocated array of integers.
    If the given node is not a centroid print: Returned array will have only one
        element, this element will be 0.
    Else,
    0th element is the number of nodes in the connected component (including the
        centroid)
    The other elements are nodes in that connected component sorted in ascending order
        .
    For example the 1st element of the returned array is the node with the smallest
        value in that connected component.
    Let's name the returned array as returned_array,
    If the length of the returned array is not equal to returned_array[0]+1, you may
        get a segmentation fault
    */
    int* nodes_in_connected_component_with_centroid (int centroid_node);

    /* ADD MORE PUBLIC FUNCTIONS IF NEEDED */
};
#endif
```

Here is an example test program that uses this class and the corresponding output. Your implementation should use the format given in the example output to display the messages expected as the result of the defined functions.

Example test code:

```
#include "Graph.h"
#include<iostream>

using namespace std;

int main() {
    // Define edges
    Edge edges_array[10] = {
        {0, 1, 3},
        {0, 3, 3},
        {0, 4, 1},
        {2, 1, 3},
        {2, 4, 1},
        {2, 3, 3},
        {1, 4, 1},
        {4, 3, 1},
        {5, 6, 12},
        {5, 7, 18}
    };
```

```
    int num_nodes = 8;
    int num_edges = 10;

    // Create graph
    Graph graph(edges_array, num_nodes, num_edges);

    for(int i = 0; i < num_nodes; i++)
    {
        int *distances = graph.dijkstra(i);
        for(int j =0; j < num_nodes; j++)
        {
            if(distances[j] == -1)
                cout<<"It is not possible to reach from node "<<i<<" to node "<<j<<"."
                    <<endl;
            else
                cout<<"Distance from node "<<i<<" to node "<<j<<" is "<<distances[j]<<"
                    ."<<endl;
        }
        delete distances;
    }

    int* centroids = graph.get_centroids();
    int num_centroids = centroids[0];
    cout<<"Number of connected components is: "<<num_centroids<<endl;

    for(int i = 1; i <= num_centroids; i++)
    {
        cout<<"Nodes in the connected component with centroid "<<centroids[i]<<" is:"<<
            endl;
        int *nodes = graph.nodes_in_connected_component_with_centroid(centroids[i]);
        for (int j = 1; j<= nodes[0];j++)
            cout<<nodes[j]<<" ";
        cout<<endl;
        delete nodes;
    }

    for(int i = 0; i < num_nodes; i++)
    {
        cout<<"Centroid node of the connected component containing the node "<<i<<" is:
             node "<<graph.get_centroid_of_connected_component_containing(i)<<endl;
    }
    return 0;
}
```

Output of the example test code:

```
Distance from node 0 to node 0 is 0.
Distance from node 0 to node 1 is 2.
Distance from node 0 to node 2 is 2.
Distance from node 0 to node 3 is 2.
Distance from node 0 to node 4 is 1.
It is not possible to reach from node 0 to node 5.
It is not possible to reach from node 0 to node 6.
It is not possible to reach from node 0 to node 7.
Distance from node 1 to node 0 is 2.
Distance from node 1 to node 1 is 0.
```

```
Distance from node 1 to node 2 is 2.
Distance from node 1 to node 3 is 2.
Distance from node 1 to node 4 is 1.
It is not possible to reach from node 1 to node 5.
It is not possible to reach from node 1 to node 6.
It is not possible to reach from node 1 to node 7.
Distance from node 2 to node 0 is 2.
Distance from node 2 to node 1 is 2.
Distance from node 2 to node 2 is 0.
Distance from node 2 to node 3 is 2.
Distance from node 2 to node 4 is 1.
It is not possible to reach from node 2 to node 5.
It is not possible to reach from node 2 to node 6.
It is not possible to reach from node 2 to node 7.
Distance from node 3 to node 0 is 2.
Distance from node 3 to node 1 is 2.
Distance from node 3 to node 2 is 2.
Distance from node 3 to node 3 is 0.
Distance from node 3 to node 4 is 1.
It is not possible to reach from node 3 to node 5.
It is not possible to reach from node 3 to node 6.
It is not possible to reach from node 3 to node 7.
Distance from node 4 to node 0 is 1.
Distance from node 4 to node 1 is 1.
Distance from node 4 to node 2 is 1.
Distance from node 4 to node 3 is 1.
Distance from node 4 to node 4 is 0.
It is not possible to reach from node 4 to node 5.
It is not possible to reach from node 4 to node 6.
It is not possible to reach from node 4 to node 7.
It is not possible to reach from node 5 to node 0.
It is not possible to reach from node 5 to node 1.
It is not possible to reach from node 5 to node 2.
It is not possible to reach from node 5 to node 3.
It is not possible to reach from node 5 to node 4.
Distance from node 5 to node 5 is 0.
Distance from node 5 to node 6 is 12.
Distance from node 5 to node 7 is 18.
It is not possible to reach from node 6 to node 0.
It is not possible to reach from node 6 to node 1.
It is not possible to reach from node 6 to node 2.
It is not possible to reach from node 6 to node 3.
It is not possible to reach from node 6 to node 4.
Distance from node 6 to node 5 is 12.
Distance from node 6 to node 6 is 0.
Distance from node 6 to node 7 is 30.
It is not possible to reach from node 7 to node 0.
It is not possible to reach from node 7 to node 1.
It is not possible to reach from node 7 to node 2.
It is not possible to reach from node 7 to node 3.
It is not possible to reach from node 7 to node 4.
Distance from node 7 to node 5 is 18.
Distance from node 7 to node 6 is 30.
Distance from node 7 to node 7 is 0.
Number of connected components is: 2
Nodes in the connected component with centroid 4 is:
```

```
0 1 2 3 4
Nodes in the connected component with centroid 5 is:
5 6 7
Centroid node of the connected component containing the node 0 is: node 4
Centroid node of the connected component containing the node 1 is: node 4
Centroid node of the connected component containing the node 2 is: node 4
Centroid node of the connected component containing the node 3 is: node 4
Centroid node of the connected component containing the node 4 is: node 4
Centroid node of the connected component containing the node 5 is: node 5
Centroid node of the connected component containing the node 6 is: node 5
Centroid node of the connected component containing the node 7 is: node 5
```

IMPORTANT NOTES:
Do not start your homework before reading these notes!!!

NOTES ABOUT IMPLEMENTATION:

1. You ARE NOT ALLOWED to modify the given parts of the header file. You MUST use adjacency matrix in your implementation. You will get no points if you use dynamic or fixed-sized arrays or any other data structures such as vectors/arrays/lists/sets/maps from the standard library.
2. Moreover, you ARE NOT ALLOWED to use any global variables or any global functions.
3. Output message for each operation MUST exactly match the format shown in the output of the example code. Otherwise, you cannot receive points.
4. Using STL (Standard Template Library) is strictly forbidden. If you use STL, you get a direct 0.
5. Cheating and any type of plagiarism is strictly forbidden. If you cheat and/or plagiarise, you get a direct 0.

NOTES ABOUT SUBMISSION:

1. In this assignment, you must have separate interface and implementation files (i.e., separate `.h` and `.cpp` files) for your class. Your class names MUST BE `Edge`, `Graph`. Your file names MUST BE `Edge.h`, `Graph.h`, `Graph.cpp`,. You will not submit `Edge.h`, since you should not modify anything in this file But you will submit `Graph.h`.
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. In the "edit" section of VPL, you can run and evaluate your code with a few example test cases. You can see whether you pass or fail them. We recommend you to use it. It may help you pinpoint the errors in your codes. Proposed grade shown in VPL may not be same as your final grade of this assignment. Since we test your codes with not only example test cases but also some additional test cases in the grading phase
4. This assignment is due by 23:59 on June 10, 2024. You should upload your work to the corresponding VPL activity in ODTUCLass before the deadline. No hardcopy submission is needed.
5. This homework will be graded by your TA **Burak Ferit Aktan** (aktan@ceng.metu.edu.tr). Thus, you may ask your homework related questions directly to him. There will also be a forum on ODTUClass for questions.