CS 202 Fundamental Structures of Computer Science II
Assignment 5 – Graphs

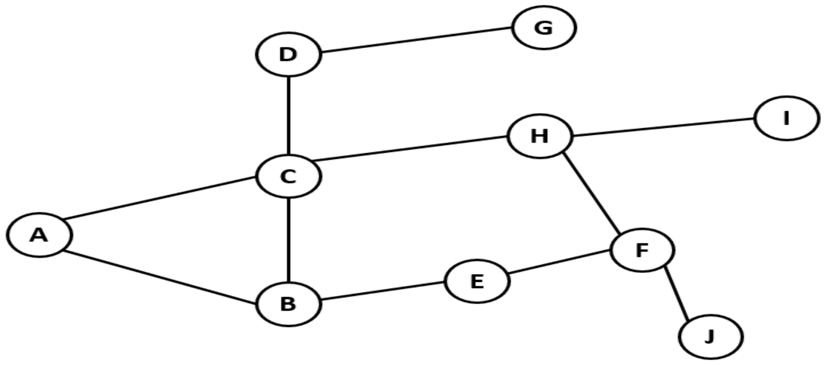
Assigned on: December 14 2016
Due Date: December 30 2016

Question-1 (20 points)

For the graph given below, give the sequence of vertices when they are traversed starting from *vertex A* using

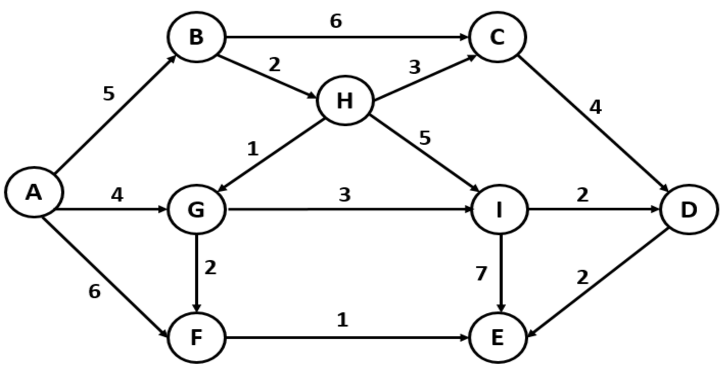
- The depth first traversal algorithm
- The breadth first traversal algorithm

In your solution, for a vertex, use the lexicographical order to visit its neighbors.



Question-2 (20 points)

- For the following weighted directed graph, find the shortest paths from *vertex A* to all other vertices using Dijkstra’s shortest path algorithm. Show all steps of Dijkstra’s algorithm.
- For the undirected version of the graph, find the minimum spanning tree using Prim’s algorithm. Start at vertex “I”. Show all steps of the algorithm.



<u>Question-3 (60 points)</u>

Programming Assignment

In this question, you are asked to implement a simple host network which represents host (machine) communication relationships. In this implementation, you will represent this communication network by a graph and answer the queries, each of which corresponds to calling a member function, on this graph. Draft of your public class **HostNet** is given below and name of the class must be **HostNet.** This class must include public member functions to conduct required tasks. These public member functions will be used to test your implementation.

```
#include <string>
using namespace std;
class HostNet{
public:
      HostNet(const string hName);
      HostNet(const HostNet& hNet); // copy contructor
      ~HostNet(); // destructor
       void listCommunicationsLowerThan(const string aname, const int
cost);
       void listCommunicationsOnHubHost(const string hname);
       void findConnectedComponents();
       void displayMostCentralHosts();
       void displayAverageClusteringCoefficients();


      // ...
      // define other public member functions here,
      // if you have any
private:
      // ...
      // define your data members here
      // define private member functions here, if you have any

};
```

The interface for the class must be written in a file called **HostNet.h** and its implementation must be written in a file called **HostNet.cpp**. You can define additional public and private member functions and private data members in this class. You can also define additional classes in your solution. The details of the member functions are as follows:

- **HostNet(const string hName);**

  The default constructor loads a host network from an input file called **hName**. The first row of this file indicates the number of hosts in the network. Subsequent lines of file include information of host and their connections. Connections between hosts are bidirectional, that is, if there is a bus from O to D, then there also exists a bus from D to O. Information in the lines contains host id, host name, number of connections, and ids of connected hosts tokens separated by white spaces.

In this assignment, you may assume that the contents of the input file are always valid. You may also assume that ids and names of the host are unique. Note that, if the input file **hName** does not exist, then the default constructor creates an empty host network.

The following table shows an example input file for the defined host network. This file contains 16 hosts, as indicated in its first line. For example, the fourth line of this file indicates that the host with id 2 has two connections to hosts with ids: 0 and 1.

```
16
0 m0 2 1 2
1 m1 3 0 2 3
2 m2 2 0 1
3 m3 4 1 4 5 6
4 m4 1 3
5 m5 3 3 7 10
6 m6 1 3
7 m7 3 5 8 9
8 m8 2 7 9
9 m9 3 7 8 10
10 m10 2 5 9
11 m11 2 12 13
12 m12 2 11 13
13 m13 4 11 12 14 15
14 m14 1 13
15 m15 1 14
```

- **void listCommunicationsLowerThan(const string aname , const int cost);**

  It outputs the hosts that are reachable with cost lower than “cost” from host “hname”. If this given host does not take place in the host network, give a warning message. See the output example below for the format. You may assume that the names are unique within the host network.

- **void listCommunicationsOnHubHost(const string hname);**

  It outputs the routes (<o, d> tuples) for which there exist no direct connection between <o> and <d>, but <d> can be reached from <o> using host “hname” as a hub host. You may consider only first level neighborhood of “hname”. If this given host does not take place in the host network, give a warning message. Similarly, you may assume that the names are unique within the host network.

- **void findConnectedComponents();**

  This member function determines whether given graph has more than one connected components or not. If graph includes only one connected component than, you should give this as an information message, otherwise you should display number of nodes of connected components.

- **void displayMostCentralHosts();**

  It should calculate betweenness centrality score of each vertex in graph and it should output the most central host that has the highest betweenness centrality score for each connected component separately.

  "**Betweenness centrality** is an indicator of a node's centrality in a network. It is equal to the number of shortest paths from all vertices to all others that pass through that node" (from wikipedia).

  The betweenness centrality of vertex *v* can be calculated as follows.

$$bc(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

  where σ_{st} is the total number of shortest paths from vertex s to vertex t and $\sigma_{st}(v)$ is the number of those paths that pass through vertex v.

- **void displayAverageClusteringCoefficients();**

  It calculates and displays the average clustering coefficient of each connected component within the network. The clustering coefficient of a vertex quantifies the connectivity information in its neighborhood. The clustering coefficient $CC\ (v)$ of *vertex v* is defined as

$$CC(v) = \frac{2 * E(v)}{d(v).[d(v) - 1]}$$

  where $d(v)$ is the number of neighbors of *vertex v* and $E(v)$ is the number of existing edges between these neighbors. The clustering coefficient of a vertex with 0 or 1 neighbor is 0. The average clustering coefficient of a connected component is the mean of the clustering coefficients computed for every vertex in this connected component.

  Below is an example test program that uses this class and the corresponding output. This test program uses the host network illustrated above. Assume that the name of the input file is "hName". We will use a similar program to test your solution so make sure that the name of the class is **HostNet**, its interface is in the file called **HostNet.h**, and the required member functions are defined as shown above.

```
#include "HostNet.h"
#include <iostream>
using namespace std;

int main(){
      HostNet HN("hName");
      HN.listCommunicationsLowerThan("m3", 2);
      HN.listCommunicationsLowerThan("m23", 2);
      cout << endl;
      HN.listCommunicationsOnHubHost("m1");
      cout << endl;

      HN.findConnectedComponents();
      cout << endl;

      HN.displayMostCentralHosts();
      cout << endl;

      HN.displayAverageClusteringCoefficients();
      cout << endl;

      return 0;
}
```

The output of this program will be as follows. Following output is not exact output for the input file. It is for sample output format and may include incomplete results or wrong results considering the input file.

```
From "m3" 3 hosts are directly reachable with cost lower than 2:
m1, m4, m5, m6.

"m23" does not exist in the host network.

If m1 is considered as hub these routes are possible:
<m0, m3>
<m2, m3>

There are two connected components:
For component 1: 11 nodes
For component 2: 5 nodes

Most central node in component 1: m3
Most central node in component 2: m13

Average clustering coefficient for component 1: 2.24
Average clustering coefficient for component 1: 0.78
```

Code Format and Notifications

You have to follow the following instructions about the format, programming style and general layout of your program.

- You can use the codes provided in your textbook or the lecture slides. However, you cannot use any external graph implementation such as STL's in your code. You ARE NOT

ALLOWED to use any codes from somewhere else (e.g., from the internet, other text books, other slides ...).

- Don’t forget to write down your id, name, section, assignment number or any other information relevant to your program in the beginning of <u>every file</u> that you are submitting.
- Don’t forget to write comments at important parts of your code.
- Be careful about indentation.
- Pay attention to these instructions, otherwise you may lose some points even though your code has no error.
- Keep all the files before you receive your grade.
- **IMPORTANT: Although you may use any platform and any operating system in implementing your algorithms and obtaining your experimental results, your code should work in a Linux environment with the g++ compiler. We will test your codes in a Linux environment. Thus, you may lose a significant amount of points, if your C++ code does not compile or execute in a Linux environment. Therefore, we recommend you to make sure that your program compiles and properly works on Linux environment with the g++ compiler before submitting your assignment.**
- Before 23:55 on due date as specified above upload your solutions using the online submission form existing in the Moodle system, attaching one zipped file that contains
  - ✓ ***Section_ID_SurnameName.pdf,*** the file containing the answers to Questions 1 and 2 (do NOT send photos of your solution, type it if it is possible, scan if it is not), and the sample output of the program in Question 3 for the provided main function,
  - ✓ HostNet.cpp and HostNet.h, the files containing the C++ codes, and the files for your additional classes, if you implemented any, and
  - ✓ readme.txt, the file containing anything important on the compilation and execution of your program in Question 3.
  - ✓ Do not submit any code containing the main function, for Question 3. We will write our own main function to test your code.
  - ✓ Do not change the prototypes of the member functions given above, for Question 3. We will call these functions, as they are, to test your code.
  - ✓ In the implementation of Question 3, you should not have any memory leaks.
  - ✓ Do not forget to put your name, student id, and section number, in all of these files. Well comment your implementation.
- Late submissions will not be graded.
- This homework will be graded by your TA Mehmet Güvercin (mehmet dot guvercin at bilkent edu tr). You may contact him for further questions.

DO THE HOMEWORK YOURSELF. PLAGIARISM AND CHEATING ARE HEAVILY PUNISHED!!!