Langara College
CPSC 2150
Assignment #10: Graphs

Assignment due with Brightspace at 11:50pm on April 7

Read chapter 8.1, 8.2, 8.3, 8.4, 8.5 of the textbook

Purpose

- To implement graph algorithms using the *adjacency list* or *list of neighbours* graph representation for a simple undirected graph.
- To implement the algorithms from scratch, not simply by modifying existing code.
- To integrate code into an existing program, understanding other people's code.

Implement a class Graph that provides the following functionality:

- creates an empty graph
- reads in data from an input stream and builds a graph
- outputs the graph
- returns how many vertices are in the graph
- determines if the graph is connected using a breadth first search
- determines if the graph has at least one cycle
- prints (lists) all the connected components of the graph:
  a connected component is a subset of the vertices that are connected
  a singleton vertex by itself is considered to be a connected component
  A graph that is not connected could have connected components.

About your implementation:

- you must use for the adjacency list (also called list of neighbours)
  - a dynamically allocated array of n pointers let's call it G
  - n (possibly empty) linked lists of nodes
  - each node has a neighbouring vertex and a pointer to the next node. So G[i] will be the linked list of neighbours of the vertex i.

  Do NOT use the STL vector[1].

  Do not use variable length arrays as these are not standard ANSI C++:

[1] Yes, yes, an STL vector could have been used but we still need practice in programming memory deallocation, destructors and in preventing dangling pointers

- overload the input operator >> so that the graph can be constructed using data obtained from an input stream
  After the graph has been read and built, note that you are not going to support the functionality of adding or removing an edge or vertex.
  Before the graph data is read using the operator >>, the previous graph is destroyed (and memory is deallocated).

- overload the output operator << to output the graph
  the format for the output is not specified

- free up memory that is no longer needed and program the copy constructor, destructor and the overloaded assignment operator

- You MUST do a breadth first search when determining if a graph is connected.

- I do not recommend implementing the cycle detection algorithm as described in the book… think of how else you could do it. There are several ways.

- use the Graph.h provided for you and expand the 'private' part of the class: use the documentation of the 'public' part

- Do not use the STL with the exception of these (which you may or may not need): std::string, std::stack, std::queue, std::priority_queue

Example

```
7
0 1
1 2
2 3
2 4
4 3
this graph has a cycle, is NOT connected and has as
components
0 1 2 4 3
5
6
```

the first 7 indicates that there are 7 vertices in the graph: the vertices are labelled 0 1 2 3 4 5 6
so in general for n vertices, the vertices are labelled 0 1 … n-1

0 1 is the undirected edge from vertex 0 to vertex 1 and so on

you can read with >> into an int until the stream fails and that's why you can put comments at the bottom of the file that will be ignored. In this case I have put that the graph does have a cycle, is not connected and that the connected components are the set of vertices {0, 1, 2, 4, 3}, the set {5} and the set {6}.

Except for the optional comments at the bottom of the file, use this file format for your program namely, the graph file will have the number of vertices n followed by edges as pairs of vertices labelled 0 1 … n-1

Application File

An application file is provided for you: it is in the file called doGraph.cpp
It is a simple command driven (menu style) program that tests your Graph class. See the documentation on the application file.

Your code must comply with the interface of doGraph.cpp given in Graph.h

To submit as Assignment #10 as a single compressed zip file

i. the source code
   a. Graph.h expand big time
      if you declare a node as a class, submit the necessary files and fix the Makefile provided
   b. Graph.cpp implement the functions
      add the missing code and other functions and variables

   submit as reference
   c. doGraph.cpp Commands.cpp Commands.h
   d. a Makefile to compile and link doGraph.cpp in C++17
      a Makefile has been provided for you
   e. the input files g0.txt g1.txt g2.txt g3.txt g4.txt g5.txt g6.txt g7.txt g8.txt g9.txt and the file input.txt

ii. A file called README.txt (if needed)
   a. indicate if you did implement the bonus part
   b. list parts of the assignment <u>not</u> done (if applicable)
   c. list bugs if known

If you do not do the bonus part, please make sure that your code still compiles and links.

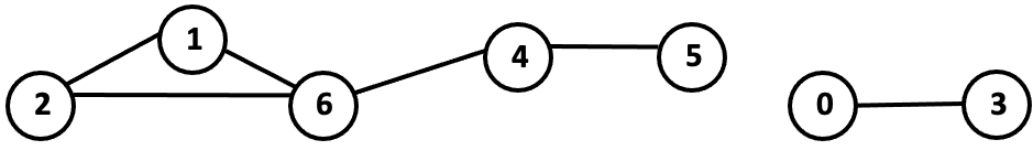
Bonus

Given an undirected simple graph G with n vertices, write a C++ function pathLength that returns the length of a path from the starting vertex vStart in G to the destination vertex vDest in G.

If there is no path from vStart to vDest, pathLength should return -1

If vStart is equal to vDest, pathLength should return 0

Example: Given is the graph G with 7 vertices labeled 0, 1, 2, 3, 4, 5, 6 and the undirected edges: edge (2,6), edge (6,4), edge (1,2), edge (1,6), edge (4,5), edge (3,0).



the call to pathLength with vStart=2 and vDest=4, could correctly return 3 or it could correctly return 2