Basic Graph Algorithms

Out: 3/12
Due: 3/25 by 11:59 PM

Learning Objectives

- Augmenting a Weighted Digraph ADT
- Using Graph Traversal Algorithms
- Implementing a Shortest Path Algorithm
- Creating the Complement of a Graph
- Generating a $\{-1, 0, 1\}$ Incidence Matrix

There are many problems in computer science that can be modeled using graphs. This project involves the augmentation of an adjacency list implementation of a weighted digraph ADT and the completion of a menu-driven application that uses the ADT. The project involves writing both ADT and client functions (methods) to perform various tasks. Some of the functions (methods) have already been written for you; so you will simply have to properly call them. You will implement other functions (methods). After this project, menu options 1-4 should work. A simplifying assumption is that the weights on the edges are non-negative real numbers.

The program will have the following text-based user interface:

```
  BASIC WEIGHTED GRAPH APPLICATION
=============================================
[1] Incidence Matrix of G
[2] Floyd's Shortest Round Trip in G
[3] Postorder DFS Traversal of G Complement
[4] BFS Traversal of G Complement
[5] Check whether G is Bipartite
[6] Topological Ordering of V(G)
[7] Prim's Minimum Spanning Tree in G
[0] Quit
=============================================
```

Definition 1. The **complement** or **inverse** of a digraph D, denoted $\bar{D}$, is a digraph on the same set of vertices such that if $(u, v) \notin E(D)$ then $(u, v) \in E(\overline{D})$, vice versa.

Definition 2. The **incidence** of a digraph is a $\{-1, 0, 1\}$-matrix which has a row for each vertex and column for each edge, and $(u, e) = 1$ if e is an out-directed edge from u and $(v, e) = -1$ if e is an in-directed edge that is incident with v and all other entries in the column label e is 0,

When option 1 is selected, your application should generate the incidence $\{-1, 0, 1\}$-matrix of the input digraph and display the number of edges in both the digraph and its complement. This option will require your completion of the *isEdge* and countEdges member functions (methods) and non-member *complement* functions (methods). Once you implement *isPath* member function (method) and *floyd* non-member function (method), you will be able to execute the second menu option. Both traversal functions (methods) have already been implemented for you. To get options 3 and 4 of the user interface to work properly, you will need to invoke them correctly in your application using the complement of the the input digraph and appropriate lambda functions. The executable file is **GraphDemo**. The input weighted digraph file will be a variation on the DIMACS network flow format described below. DIMACS is the Center for Discrete Mathematics and Theoretical Computer Science based at Rutgers University.

- **Comments.** Comment lines give human-readable information about the file and are ignored by programs. Comment lines can appear anywhere in the file. Each comment line begins with a lower-case character **c**.

  ```
   c This is an example of a comment line.
  ```

- **Problem line.** There is one problem line per input file. The problem line must appear before any node or edge descriptor lines.

  ```
  p NODES EDGES
  ```

  The lower-case character `p` signifies that this is the problem line. The `NODES` field contains an integer value specifying $|V|$, the number of vertices in the graph. The `EDGES` field contains an integer value specifying $|E|$, the number of edges in the graph.

- **Node Descriptors.** All node descriptor lines must appear before all edge descriptor lines.

  ```
  n ID LABEL
  ```

  The lower-case character `n` signifies that this is a node descriptor line. The `ID` field gives a node identification number, an integer between 1 and $|V|$. The `LABEL` field gives a string which serves as an alternate label for the vertex.

- **EDGE Descriptors.** There is one edge descriptor line for each edge in the graph. The edge descriptor line will be formatted as:

  ```
  e SRC DST WEIGHT
  ```

  The lower-case character **e** signifies that this is an edge descriptor line. For a directed edge (v, w), the `SRC` field gives the identification number for the source vertex v, and the `DST` field gives the destination vertex w. For an undirected edge, (v, w) and (w, v) refer to the same edge so only one edge descriptor line appears and on that line the end points of the edge are written in lexicographical order. Identification numbers are integers between 1 and $|V|$. The `WEIGHT` field contains $cost(v, w)$.

The input file name is entered as a command line argument. For example, to run your application on a graph file called cities1.wdg, enter *cities1.wdg* as a command line argument. The assumption is that file is in DIMACS format so no validation is done on the input file. The *readGraph* function (method) has already been implemented for you and it reads the input file and creates a *Graph* instance. The files *cities1.wdg* and *cities2.wdg* are weighted digraphs. Any digraph file may be used to test menu options 1-4.

Submitting Your Work

1. All source code files you submit must have a header with the following:

```
/**
* Describe the purpose of this file
* @author Programmer(s)
* @since 9999-99-99
* Course: CS3102.01
* Programming Project #: 3
* Instructor: Dr. Duncan
* @see the list files, if any, that this file references
*/
```

2. Verify that your code has no syntax error and that it is ANSI/ISO C++11 or Java$^{\text{TM}}$JDK 8 compliant prior to uploading it to the drop box on Moodle. Be sure to provide documentation, where applicable, for the additional methods that you have been asked to write. Also, add your name after the @author tag when you augment the starter code that I have provided. Put the last date modified in the header comments for each source code file.

3. Enclose your source files-

   (a) for Java programmers, **GraphAPI.java**, **Graph.java**, **GraphDemo.java** and **City.java**.

   (b) for C++ programmers, **Graph.h**, **Graph.cpp**, **GraphDemo.cpp** and **City.h**.

   - in a zip file. Name the zip file *YOURPAWSID_proj03.zip*, and submit your project for grading using the digital drop box on the course Moodle.