**Note:** All additions are below the original specifications (i.e., the original text is unchanged), with the exception of one (marked) deletion.

**--------------**

**Algorithms**

Fall 2015 **Graphs Warm-up**

Dr. Gurka preliminary development

August 27

This exercise focuses on developing a graphs package that you can use in later projects. Develop it thinking of yourself as the future user, in the sense that you will be writing programs that need to use graphs, and will treat this code as a library resource (similar to Java’s API). Secondarily, you will become well-versed in the graph data structures and operations. Thirdly, you will refine your program development skills.

Start this exercise in class today, and continue to work at home. The completion goal is two weeks. There is nothing to hand in, but you will submit two or three progress reports along the way.

Use you prior experience developing projects to have a well-organized approach for design, development, and test. Keep track of your process as you go, including time spent. Structure this information as an informal log to be turned in with the progress reports.

You may work together, but ensure that you fully understand each component of your system.

----

You will create two packages, one using an adjacency matrix and one using an adjacency list. The first use of the packages will be to run performance analysis experiments comparing the different implementations. Later you’ll use the packages as the basis for various graph problems.

Everyone will create test plans, to be shared. The format for graph input and output will be developed in class.

-----

-----

Graph functions to implement:

* hasEdge, addEdge, deleteEdge
* isSparse (use 15%), isDense (use 85%)
* isConnected, isFullyConnected
* ~~insertEdge, deleteEdge~~ **(duplicate – deleted)**
* readGraph, printGraph (both text-based, format to be decided in class)

-----

Additional specifications.

* data file format will be correct, no error-checking needed
* graphs in the data file will be valid
* no edges with only one vertex
* hasEdge asks whether a specific edge exists in the graph, not whether the entire graph has any edges
* new graph functions: addVertex, deleteVertex
* weights, if any, will be positive floats
* vertex names will be a single uppercase letter (note the resulting limit of 26 vertices in a graph)
* adding a duplicate vertex has no effect on the graph (i.e., nothing is deleted) & returns false
* test files should include your name and a brief description of the case being tested (type of graph, order of operations, etc.) – what is the design of this specific test case?
* name your test file <your first name><k>.txt (Judy3.txt) for easy swapping
* edges will not be repeated in the graph edge list
* addEdge can overwrite an existing edge (applicable only to weighted graphs)

-----

-----

Required test file format (they will be exchanged).

Notes

* ‘\*’ as the first character in the line means a comment
  + several lines of comments at the beginning (see above)
  + comments can also occur between test cases for specific info on one test case
* file format is perfect (no error checking)
* each test file contains one graph and the operations on it
* edges listed in the file are in no particular order
* if the graph is directed, the edge goes from the first vertex listed to the second
* an empty graph and a graph composed of a single node are both considered connected
* an empty graph has no lines between “begin” and “end”
* all text except comments and vertex names will be lowercase, except camel case for functions (see list???)
* output: the program should echo the original graph, in the same form (checks correct input and documents output for easy checking)
* the number of edges is unknown
* edge data and vertex data is space separated
  + example: <vertex 1><space><vertex 2><space><weight>
* no edges that start and end at the same vertex
* the graph is not a multi-graph (2 or more edges between 2 vertices); exception: in a weighted graph, there can be two edges: one from the first vertex to the second, and one from the second vertex to the first – they need not have the same weight
* no blank lines at the beginning or end of the data file

-----

Function names, to be spelled exactly as shown (including case) for inclusion in the data file:

* hasEdge
* addEdge
* deleteEdge
* isSparse
* isDense
* isConnected
* isFullyConnected
* readGraph
* printGraph

-----

-----

General file format.

<overall header comments>

<weighted or unweighted> (actual keywords)

<directed or undirected> (actual keywords)

begin

<list of vertices>

<edge 1>

<edge 2>

:

<edge n>

end

<function 1>

<correct result 1>

<function 2>

<correct result 2>

:

<function m>

<correct result m>

-----

Sample test file.

\* Graphs test file by J. Gurka

\* File: Judy7.txt

\* This file tests adding and deleting edges in a sparse graph, including

\* duplicate edges being added and non-edges being deleted.

weighted

undirected

begin

A F B C D

A B 2.3

C F 1.0

A D 6.2

end

hasEdge B Q

false

isSparse

true

deleteEdge C F

true (success)

-----