Homework 8

Out: 11.24.20
Due: 12.8.20

1. [Single-Source Shortest Path, 10 points]
Design an efficient algorithm to list the vertices along a cycle of a weighted, directed, graph G. Analyze the runtime of your algorithm.

2. [Single-source Shortest Path, 10 points]
Design an efficient algorithm that outputs the overall number of paths that exist in a given directed acyclic graph. Analyze the runtime of your algorithm.

3. [Single-source Shortest Path, 10 points]
Design an efficient algorithm that receives as input a weighted graph G with non-negative edge weights, a source vertex s, and d and π arrays for the graph. The algorithm should verify if the provided pair of d and π arrays represent a shortest-path tree for the given graph, starting at vertex s. Analyze the runtime of your algorithm.

4. [All Pairs Shortest Path, 10 points]
Why does Johnson's algorithm add vertex s to V in its first step? Why not just use any existing vertex in the input graph? Explain, and give an example of a directed graph for which Johnson's algorithm would not work without adding a vertex s to V in the initial step.

5. [All Pairs Shortest Path, 10 points]
If the input graph to Johnson's algorithm has no negative edge weights, what is the relationship between the original edge weights and the re-weighted weights? Explain.

6. [Graphs, 50 points]
The *Graph* class, specified in the header file *Graph.h*, represents a directed, weighted graph. Also provided is a sample *main.cpp* file, and a sample input file *graph.txt*. These demonstrate how we will test your code.

The input file is organized as follows:

- The first line indicates the (# of vertices) (# of edges) (type of graph)
- The (# of vertices) indicate the names of the vertices, i.e. if the (# of vertices) is 4, then the names of the vertices are {0, 1, 2, 3}.
- The rest of the lines indicate the (source vertex) (end vertex) (weight).

a. Implement the Graph class in *Graph.cpp*, according to the specifications in *Graph.h*, which are repeated here.

The graph class contains the following private members:
- set<int> vertices – The set of vertices of the graph.
- set<directedEdge> edges – The set of edges of the graph.

- map<directedEdge, int> weights – A mapping between edges and their weights.

The Graph class contains the following methods:
- Graph() – Constructs a graph with no edges or vertices.
- addVertex() – Adds a vertex to the graph, and returns the ID of the added vertex.
- addEdge(directedEdge newEdge, int weight) – Adds the given edge with the provided weight to the graph. The vertices of <newEdge> must currently exist in the graph.
- getNumVertices() – Returns the number of vertices in the graph.
- getWeight(directedEdge edge) – Returns the weight of the provided edge. The edge must exist in the graph.
- isEdge(directedEdge newEdge) – Returns true iff there is an edge in the graph with the same vertices as newEdge.
- print() – Prints a human-readable version of the adjacency list of the graph. Format is: vertex: adjacent_vertex_1 (weight_1) adjacent_vertex_2 (weight_2) ...
- generateGraph(string fileName) – Constructs a graph from the file with the provided name. The file format is as follows: The first line contains the number of vertices and the number of edges, separated by a space, followed by optional additional text. The graph is assumed to contain vertices numbered 0 to 'number of vertices' - 1. Each of the remaining lines contain one edge specified by the source and destination vertices followed by the weight, and separated by spaces.
  Returns the constructed graph.

Do not modify any existing code in *Graph.h* (you may add members as you see fit.) Submit your modified *Graph.h* and *Graph.cpp*. Your code should compile with the provided files on the lab computers.

Your program should be executed from the command-line with an input file name as argument. It should print out the output as in the sample below (listing source vertex to end vertices and their respective weights in parentheses).

0: 2 (13) 4 (16) 5 (8)
1: 3 (6) 5 (10)
2: 3 (14) 5 (11)
3: 4 (5) 5 (17)
4: 1 (9) 5 (7)
5:

b. Define and implement a public *modifiedDijkstra(int source)* method for the *Graph* class. This method should compute and print the weight and number of shortest paths from provided source vertex to all other vertices of the graph. You may assume that the graph is guaranteed to have non-negative weight edges, and that source is a vertex of the graph. The method should work for any source vertex.

In a comment at the top of your method, provide a brief description of how you modified Dijkstra's shortest path algorithm on a directed graph with non-negative edge weights to count the number of shortest paths from a given source s to a destination node d.

Do not modify any existing code in *Graph.h* (you may add members as you see fit.) Submit your modified *Graph.h* and *Graph.cpp*.

Your program should be executed from the command-line with an input file name as argument. It should print out the output as in the sample below.

```
0: 2 (13)  4 (16)  5 (8)
1: 3 (6)  5 (10)
2: 3 (14)  5 (11)
3: 4 (5)  5 (17)
4: 1 (9)  5 (7)
5:

Shortest paths from node 0:
Distance to vertex 1 is 25 and there are 1 shortest paths
Distance to vertex 2 is 13 and there are 1 shortest paths
Distance to vertex 3 is 27 and there are 1 shortest paths
Distance to vertex 4 is 16 and there are 1 shortest paths
Distance to vertex 5 is 8 and there are 1 shortest paths
```