CSDS 233 – Introduction to Data Structures
Assignment #6 (100 points)
Due date: December 7, 2022 11:59pm.

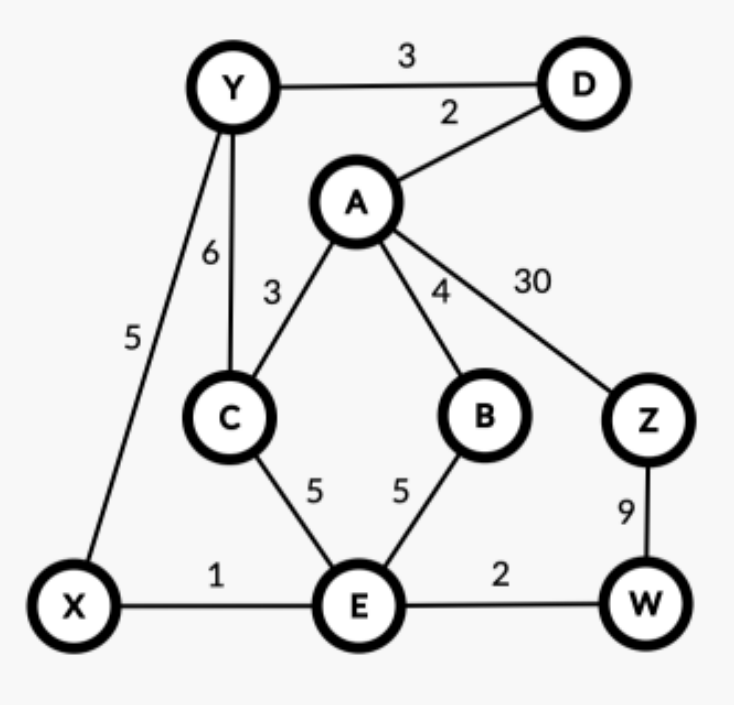
General Instruction: This assignment includes two parts, written and programming. Please write/type your answers neatly so they can be readable. Please submit a single PDF file for the written part and a zip file for the programming part before 11:59pm – December 7.

Special office hours **TBD** (Lookout for an announcement). Feel free to send an email with your questions at bsl47@case.edu and/or uxm28@case.edu

Written Exercises (50 pts)

Problem 1 (30 pts):

Given the following graph:



a) Provide a set of adjacency lists of valid movement through the graph. **(5 pts)**

b) Provide an adjacency matrix for valid movement through the graph. **(5 pts)**

c) Use Dijkstra's algorithm to find the shortest path from A to Z. For each step of the algorithm, provide the minimum cost from the source to each vertex, as well as the updated binary heap. **(10 pts)**

d) Use Prim's algorithm to find the **maximum** spanning tree from the graph above. For each step of the algorithm, provide the current state of the spanning tree. **(10 pts)**

Problem 2 (10 pts):

Consider a connected undirected graph G with N nodes. [10 pts]

a) What is the smallest possible number of edges in G? In this scenario, is this acyclic or not? Why? **(5 pts)**

b) If G is a complete graph. How many possible spanning trees G can have? Why? **(5 pts)**

Problem 3 (5 pts):

Mip is preparing for the final exam of CSDS 233 with an undirected graph, in which there are n vertices and m edges. There isn't any self-loop or multiple edges in the graph. Self-loop is defined to be an edge connecting a vertex to itself. Multiple edges are a list of edges connecting the same pair of vertices. Since the graph is undirected, (a, b) and (b, a) are multiple edges. Isolated vertex of the graph is a vertex such that its degree is 0, meaning that there is no edge connecting it to any other vertices. As she is preparing for the final exam of CSDS 233, she wants to know the following number of vertices. Please provide a description of an algorithm for each of the questions.

a) The minimum possible number of isolated vertices in Mip's graph with n vertices and m edges. **(2 pts)**

b) The maximum possible number of isolated vertices in Mip's graph with n vertices and m edges. **(3 pts)**

For example: if the graph has 4 vertices and 2 edges:

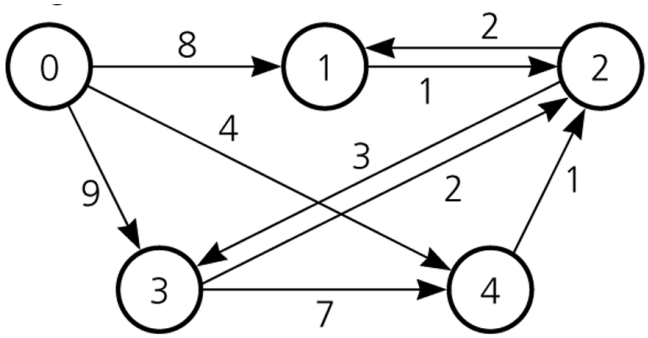
The minimum number of isolated vertices that we can have is 0, since we can have 2 edges: (1, 3) and (2, 4).

The maximum number of isolated vertices that we can have is 1, since we can have 2 edges: (2, 3) and (2, 4).

(See next page)

Problem 4 (5 pts):

Consider the weighted and directed graph below:



a) Write its adjacency matrix representation **(2 pts)**

b) Fill the below table showing the steps of Dijkstra's Shortest-Path Algorithm starting from vertex "0" **(3 pts)**

			weight				
Step	v	vertexSet	[0]	[1]	[2]	[3]	[4]

Programming Exercises (50 +10 pts)

In this assignment you will implement basic graph search algorithms. The specifications below are fairly detailed, but they do not attempt to describe every situation or consideration exhaustively. It is part of the programming exercises to decide how to handle these types of cases and do something reasonable.

Constructing undirected, unweighted graphs (35 pts)

Create a class Graph with the methods listed below. Nodes are referenced by their node names, which are Strings. The methods with boolean return types should return true if successful and false otherwise.

- boolean **addNode**(String name) - adds a node to the graph and checks for duplicates.
- boolean **addNodes**(string[] names) - adds a list of nodes to the graph and checks for duplicates.
- boolean **addEdge**(String from, String to) - adds an edge from node from to node to.
- boolean **addEdges**(String from, String[] tolist) - adds an undirected edge from node from to each node in tolist.
- boolean **removeNode**(String name) - removes a node from the graph along with all connected edges.
- boolean **removeNodes**(String[] nodelist) - removes each node in nodelist and their edges from the graph.
- void **printGraph()** - prints the graph in an adjacency list format. The nodes and their neighbors and their neighbors should be listed in alphabetical order.

Finding paths (15 pts)

Next you will implement methods to find paths between two nodes using depth-first and breadth-first search.

- String[] **DFS**(String from, String to, String neighborOrder) - returns the result, i.e. the path or a list of node names, of depth-first search between nodes from and to. The order in which neighbors are considered is specified by neighborOrder, which can be "alphabetical" or "reverse" for reverse alphabetical order. It should return an empty array if no path exists.
- String[] **BFS**(String from, String to, String neighborOrder) - as in DFS, but using the breadth-first search algorithm. If there are multiple paths of equivalent length, you only need to return one of them.
- String[] **shortestPath**(String from, String to) - uses Dijkstra's algorithm to find the shortest path from node from to node to. If there are multiple paths of equivalent length, you only need to return one of them. If the path does not exist, return an empty array.
- String[] **secondShortestPath**(String from, String to) - returns the second shortest path between nodes from and to. Again, you only need to return one path in the case of multiple equivalent results.

Your demonstration should use multiple examples to illustrate all the functionality and, in the case of DFS, how results can vary depending on the neighbor order.

Directed, weighted graphs (Extra Credit Section: 3 Pts)

Make a WeightedGraph class to work for weighted and directed graphs. Here we will assume that the graphs are both weighted and directed. As above, also write the following methods to construct graphs programmatically:

- boolean **addWeightedEdge**(String from, String to, int weight) - adds an weighted edge from node from to node to. Note that for simplicity, we will only consider integer weights. **(1 pts)**
- boolean **addWeightedEdges**(String from, String[] tolist, int[] weightlist) - adds an edge from node from to each node in tolist with the corresponding weights in weightlist. **(1 pts)**
- void **printWeightedGraph()** - prints the graph in an adjacency list format. The nodes and their neighbors and their neighbors should be listed in alphabetical order. **(1 pts)**

Shortest paths on directed, weighted graphs (Extra Credit Section: 7 Pts)

Now implement shortest path methods for directed, weighted graphs:

- String[] **shortestPath**(String from, String to) - uses Dijkstra's algorithm to find the shortest path from node from to node to. If there are multiple paths of equivalent length, you only need to return one of them. If the path does not exist, return an empty array. (**3 Pts**)
- String[] **secondShortestPath**(String from, String to) - returns the second shortest path between nodes from and to. Again, you only need to return one path in the case of multiple equivalent results. Again, return an empty array if no 2nd shortest path exists (i.e. there is no path or only one path). **(4 pts, can gain 2 pts for written explanation/theory only)**