CSCI 210/2201 Fall 2018 Dr. Amr Goneid
Assignment #7 (Bonus Assignment)

Date: Mon Dec 3, Due: Thu Dec 13, 2018

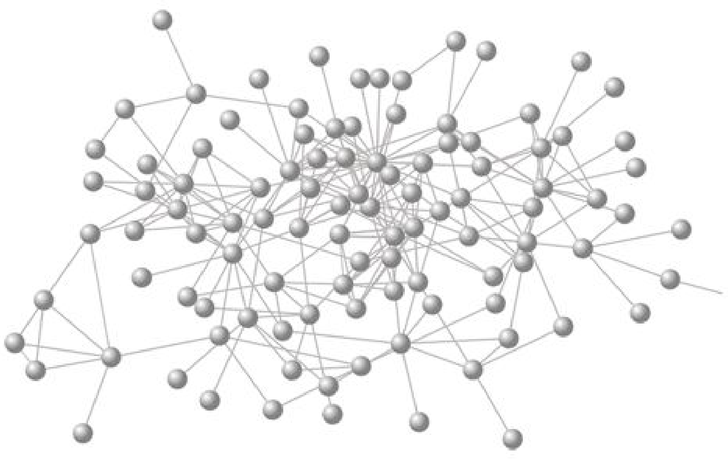
This is a bonus assignment. No submissions will be accepted after the due date.
This assignment is to be **an individual work** and will be graded as such.

The problem: Graph Traversal and Finding the Minimum Spanning Tree for a Graph

A network of dusty roads connects *N* houses in a village. During the rainy season, these roads become muddy, and there is a need to pave the roads. The problem is:

1. Starting from a given house, find the order of visiting all houses in the village.
2. Find which roads to pave such that there would be a paved path from every house to every other house and that the cost of such paving be minimum.

The houses and road network can be represented by a graph with *N* nodes representing the houses and the weights on the edges represent the distances between pairs of houses. In this case, our problem would be to traverse the graph from a given starting node and to find a ***Minimum Cost Spanning Tree (MST)*** for the given network. Such a *MST* would be connected (i.e., there is a paved path from every house to every other house) and the sum of weights of edges in that tree would be minimum (i.e., the cost of paving be minimum).



For the above problem, you are given an Excel sheet **"VillageG.xlsx"** that contains the adjacency matrix representation for the weighted graph for *N* = 20 houses. The graph is connected and the weights are all positive integers in the range 20 – 100. Zero weight represents the absence of a road, or the distance between a house and itself. The houses can be numbered (*0,1,2,…)* or (*A,B,C,…*.).

Required Implementation:

Develop a program to:

1. Traverse the given graph using the recursive Depth-First Search (***DFS)*** algorithm.
2. Implement ***Kruskal's*** algorithm for generating the ***Minimum Spanning Tree*** for the given network. Use the adjacency matrix given above to test your implementation.

Notes on the design and implementation:

- You may use an adjacency matrix representation of the graph, i.e., a 2-D array of size V_{max} by V_{max} where V_{max} is the maximum number of vertices (e.g $V_{max} = 50$).

- An edge *e = (u,v,w)* has one vertex *u*, a second vertex *v* and a weight *w* (a positive integer). It can be represented as an object of a class ***"Edge"***. You may use a 1-D array of size E_{max} to store the non-zero edges in the graph, where E_{max} is the maximum number of possible edges = $V_{max}*(V_{max}-1)/2$. The actual number of such edges in the graph will be *E* so that the edges will be stored in locations (*0 .. E-1*).

- The vertices can be given numbers (*0,1,….V-1*) where *V* is the actual number of vertices in the given graph. These numbers can be mapped to names (e.g *A,B,C…*).

- The *MST* algorithm will use a *PQ* (Minimum heap) to insert edges into the heap and then remove one edge at a time. The algorithm also uses a ***"Sets"*** class for disjoint sets in order to test for cycles in the graph.

- The zip file **"210as7F18.rar"** contains the graph file **"VillageG.xlsx"** that will be used in the assignment. The zip file also contains a test graph file **"testgraph.txt"** representing a small graph of 7 vertices. The file format is as follows:
  The first number is the number of vertices (7). The next 7 numbers represent the first row, the next 7 numbers represent the second row, etc.
  You can test your implementation using this file. The sample output for that file is also given in file **"sample output.txt".**

<u>**Output Report:**</u>
As a minimum, report the results of running your program on the graph in **"VillageG.xlsx"** in a way similar to the **"sample output.txt".**