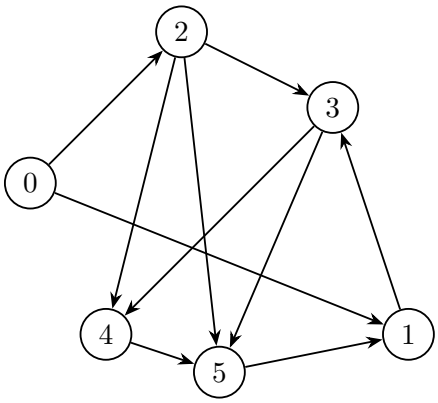
CSCI 3104 Summer 2017 - Assignment # 7

Due: July 24th 11:55pm

Be sure to justify all of your work. You must use Python to complete this assignment. All of your code should be written from scratch, i.e. no non-trivial libraries.

1. (100 points) For this assignment, you will implement simple map routing using different algorithms. First, we will be dealing with geographic vertices, in a simplified way. These vertices are points in the 2D plane (with x and y coordinates) connected by edges which are Euclidean distances. For simplicity, you may assume that $0 < x, y < 10000$.

    a) For the first part, you will read in a graph from a text file. The text file will contain (1) the number of vertices and edges, (2) the numbered vertices along with their respective x and y coordinates, (3) the edges represented as pairs of vertices. Additionally, for testing, the source and destination vertices will be included. As an example, we could represent the following graph as a text file (Note: the last line represents source and destination vertex, i.e. we want to search from vertex 2 to 1):



```
6 10

0 250 500
1 1700 200
2 800 900
3 1200 600
4 400 200
5 900 100

0 1
0 2
1 3
2 3
2 4
2 5
3 4
3 5
4 5
5 1

2 1
```

You can choose to represent the graph any way you like (e.g. an adjacency list).

b) Next, you must implement code to check whether the input graph contains a cycle.

c) In addition, you need to implement two shortest-path algorithms: (1) the algorithm we discussed in class for DAGs, and (2) Dijkstra's algorithm. Based on the result of the cycle check routine, you need to call your DAG shortest-path algorithm if the input graph is acyclic, or your Dijkstra algorithm if it contains a cycle. The output of either of these routines is (1) The source and destination vertex, (2) the total distance of the shortest path between them, and (3) the actual sequence of vertices that form the shortest path, in the correct order.

d) Finally, you must implement code that will plot the shortest path found by your shortest-path algorithm(s), along with the Euclidean distances between vertices on this path.

Note: All of your implementations must run efficiently, e.g. the DAG shortest-path algorithm should run in $\mathcal{O}(V + E)$. Provide evidence (plots) that show your implementations are sufficiently fast.

2. **Extra Credit: (10 points)** Optimize Dijkstra's algorithm so that once you have read in your graph, any shortest-path search takes sublinear time. Note that you are not allowed to simply compute and store all shortest-paths for all pairs of vertices, i.e. assume you cannot afford the space requirement for doing so. You must clearly show that your optimizations meet the time complexity criteria.