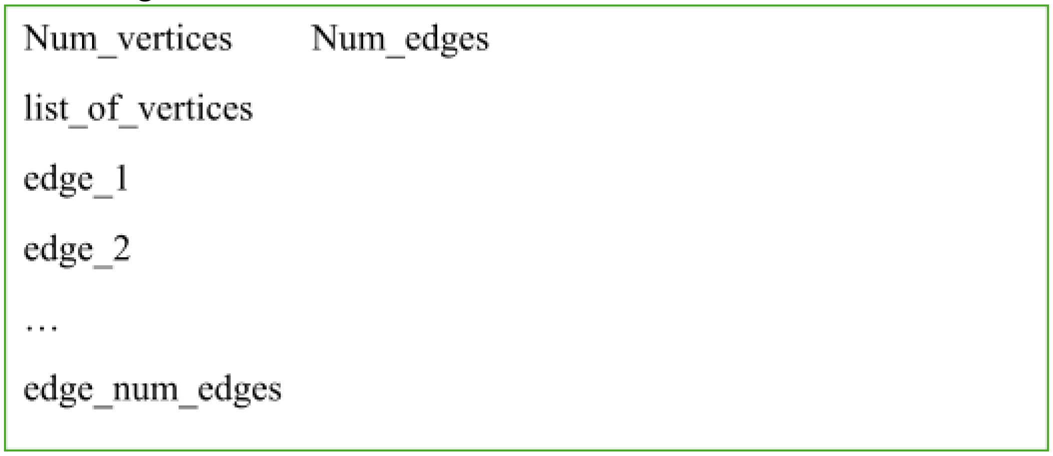
Assignment 12: Dijkstra's Algorithm

CS 301

April 15, 2024

With Dijkstra's Algorithm fresh in your minds, lets have you actually implement it in code. You will implement your code in a Python code file named Dijkstra.py .

1. First, lets have you modify your AdjacencyMatrix class from last time to implement weighted graphs. Call your modified class WeightedAdjacencyMatrix and make sure it implements the following methods:
    a. `readGraph(filepath)` reads the text file given at `filepath` and initializes a graph based on the information in the file and returns `True` if it is able to successfully read and create a graph, `False` otherwise (you may choose to print. The file is composed in the following format.:



```
Num_vertices       Num_edges
list_of_vertices
edge_1
edge_2
…
edge_num_edges
```

    The first line in the file contains two numbers; the first number is the number of vertices in the graph, and the second number is the number of edges in the graph.
    The second line contains a list of vertices separated by commas. Subsequent lines contain an edge (each on a separate line) in the graph as two vertices and the edge weight all separated by whitespace. 2 example graph files (`weightedGraph1.txt` and `weightedGraph2.txt`) are included for your testing.

   b. `addVertex(vertex)` adds a vertex to the graph indicated by the `vertex` string. If the vertex already exists, do nothing. This method returns True on success, False otherwise.
   c. `addEdge(edge)` adds an edge represented by the 3-tuple `edge` to the graph. If the edge already exists, do nothing. The method should ensure that the vertices in the edge already exist in the vertex and should return `False` if either is not present. On successful addition, the method should return `True`.
   d. `deleteVertex(vertex)` deletes a vertex from the graph indicated by `vertex`. Note that deletion on a vertex should include deletion of all edges associated with the vertex. On successful deletion, the method returns `True`, `False` otherwise.
   e. `deleteEdge(edge)` deletes the edge indicated by the 3-tuple `edge`. If the edge does not exist, return `False`. If the edge is successfully deleted, return `True`.
   f. `getNeighbors(vertex)` returns the neighbors (vertices associated with the given `vertex` with an edge) as a list of 2-tuples, where each neighboring vertex is associated with the edge weight leading to it. If the vertex has no neighbors, return an empty list. If the vertex does not exist, return `False`.
2. Implement a function Dijkstra(graph, start, end) that runs Dijkstra on the given graph, and returns a path from start to end as a list of edges (with their individual weights) and a total path weight as a 2 tuple (the first element of the tuple is the path, the second is the path weight). You may find it helpful to utilize your PriorityQueue implementation from Assignment 10 with minor modifications (such as retrieving the index of an item stored in the queue, or updating its position with an updated key.). In comments, describe the average running time of your implementation of Dijkstra's algorithm.
3. On the given weightedGraph1.txt file, run your implementation of Dikjstra's algorithm and provide the paths and weights for the following pairs:
   a. Start vertex: a, end vertex: o
   b. Start vertex: m, end vertex: d
   c. Start vertex: g, end vertex: t
   d. Start vertex: k, end vertex: p