Problem Set 5: Graph Algorithms

Handed out Friday, September 27. Due at the start of class Friday, October 4.

Problem 1. (15 points) You are working for a delivery company where you must drive trucks from a central dispatch center to various locations. The road network is modeled by a weighted digraph $G = (V, E)$, where the weight of each edge (u, v) is the distance from location u to location v in miles. Recently, the department of transportation erected toll booths in various locations. This is modeled by associating a non-negative toll value $t(u)$ with each vertex u of the digraph. Based on your level of gas consumption, you determine that you are willing to pay $\$X$ dollars in tolls in order to save $20 \cdot X$ miles. For example, if you have three possible paths, of lengths 100, 150, and 200 miles, involving \$10, \$5, and \$3 in tolls, respectively, you would prefer the second option (150 miles and \$5 in tolls).

Devise an efficient algorithm that computes the minimum cost path from the dispatch center s to every node of the digraph. As always, justify your algorithm's correctness and derive its running time. (Hint: This can be solved as fast as Dijkstra's algorithm.)

Problem 2.(15 points) You are given an undirected graph $G = (V, E)$ in which the edge weights are highly restricted. In particular, each edge has a positive integer weight of either $1, 2, ..., W$, where W is a constant (independent of the number of edges or vertices). Show that it is possible to compute the single-source shortest paths in such a graph in $O(n+m)$ time, where $n = |V|$ and $m = |E|$.
Hint: Because W is a constant, a running time of $O(W(n + m))$ is as good as $O(n + m)$.

Problem 3. (15 points) Programming - Identifying DAGs (Upload code to Moodle)

Write a Python program that reads in an adjacency matrix representing a graph, either from the command line or as input, and determines if that graph is a DAG (directed acyclic graph) or not. The output for your program should be Yes, it's a DAG, or No, it's not a DAG.

Examples:
G1 = [[0,1,1,0,1], [0,0,0,0,1], [0,1,0,1,0], [0,1,0,0,1], [0,0,0,0,0]] returns "Yes, it's a DAG."
G2 = [[0,1,0,1,0], [1,0,1,0,1], [0,1,0,1,1], [1,0,1,0,1], [0,1,1,1,0]] returns "No, it's not a DAG."

Hint: PythonReviewGraphs.py on Moodle has code to help take in a string representation of a graph like above and make it into a 2-d list

Extra Credit: If the Graph is a DAG, output a valid topological ordering of the nodes.