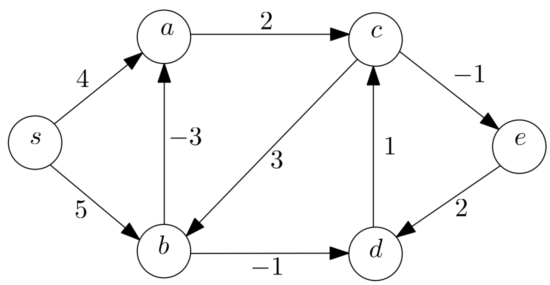
CS:3330 Homework 7, Spring 2017

Due in class on Thursday, March 30th

1. Consider the GenericSSSP algorithm on Page 4 of Lecture 21 (from Prof. Jeff Erickson's notes). (In class, I have been calling this the *Dantzig-Ford* algorithm (version 2)). As you know, when all edge weights are non-negative, if we use a min-heap priority queue implementation of the bag data structure, we get *Dijkstra's algorithm* and this runs in $O(m \log n)$ time on graphs with n vertices and m edges. So we have a pretty efficient algorithm, when edge weights are all non-negative.

   What about when some edge weights can be negative, but there are no negative cycles? Section 21.6 in Prof. Erickson's notes describes an algorithm that he calls *Shimbel's algorithm* that runs in $O(m \cdot n)$ time, solving the SSSP problem even when there are negative edge weights, provided there are no negative cycles. This algorithm is more commonly known as the *Bellman-Ford* algorithm.

   (a) Execute Shimbel's algorithm on the graph given above. As in the figure at the top of Page 7, after each phase (i) show all the $dist(\cdot)$ values, (ii) all the vertices that were in the queue during the phase that just ended, and (iii) all the edges that were relaxed in the phase that just ended.

   (b) Use induction to prove the following claim that appears in the box on Page 6: After i phases of the algorithm, $dist(v)$ is at most the length of the shortest walk from s to v consisting of at most i edges.

2. Let G be a directed, edge-weighted graph such that every edge has a weight that belongs to the set $\{0, 1, \ldots, W\}$, where W is a non-negative integer. Think of W as being a small constant, say 10. This problem asks you to take advantage of this special property of the edge weights to make Dijkstra's algorithm run *faster* than its current running time of $O(m \log n)$? Specifically, modify the implementation of Dijkstra's algorithm so that the SSSP problem can be solved in $O(n \cdot W + m)$ time for a graph with n vertices and m edges. After presenting your new algorithm present an analysis showing that its running time is $O(n \cdot W + m)$. **Hint:** Instead of using a min-heap to implement a priority queue, think about how to take advantage of the fact that $dist(\cdot)$ values are going to be integers in the range 0 through $W \cdot (n - 1)$.

3. Describe an algorithm with running time $O(m \log n)$ that computes a *maximum* spanning tree of an n-vertex m-edge graph.

4. A graph with n vertices and m edges is *dense* if $m = \Theta(n^2)$. Since Prim's algorithm runs in time $O(m \log n)$ time, on dense graphs it runs in $O(n^2 \log n)$ time. By not using a min-heap implementation of a priority queue and using a different (and very simple) data structure instead, it is possible to improve the running time of Prim's algorithm to $O(n^2)$ for dense graphs. Describe this data structure and how it is used by Prim's algorithm and then argue that with this new data structure, Prim's algorithm runs in $O(n^2)$ time.