DSC 40B - Homework 08

Due: Wednesday, May 29

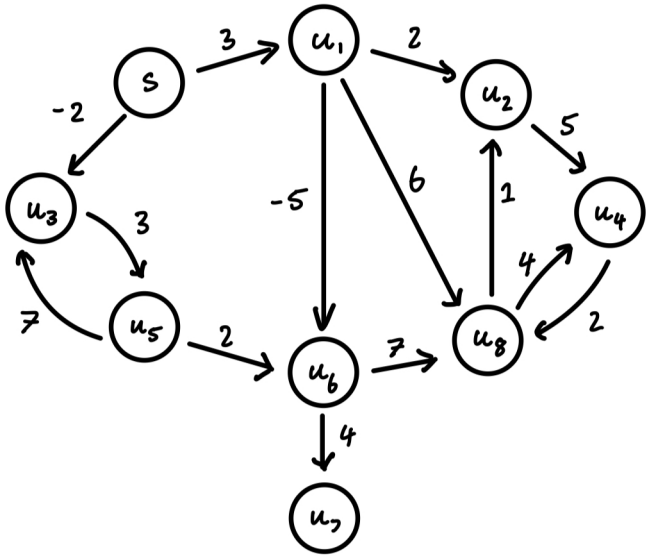
Write your solutions to the following problems by either typing them up or handwriting them on another piece of paper. Unless otherwise noted by the problem's instructions, show your work or provide some justification for your answer. Homeworks are due via Gradescope at 11:59 p.m.

Problem 1.

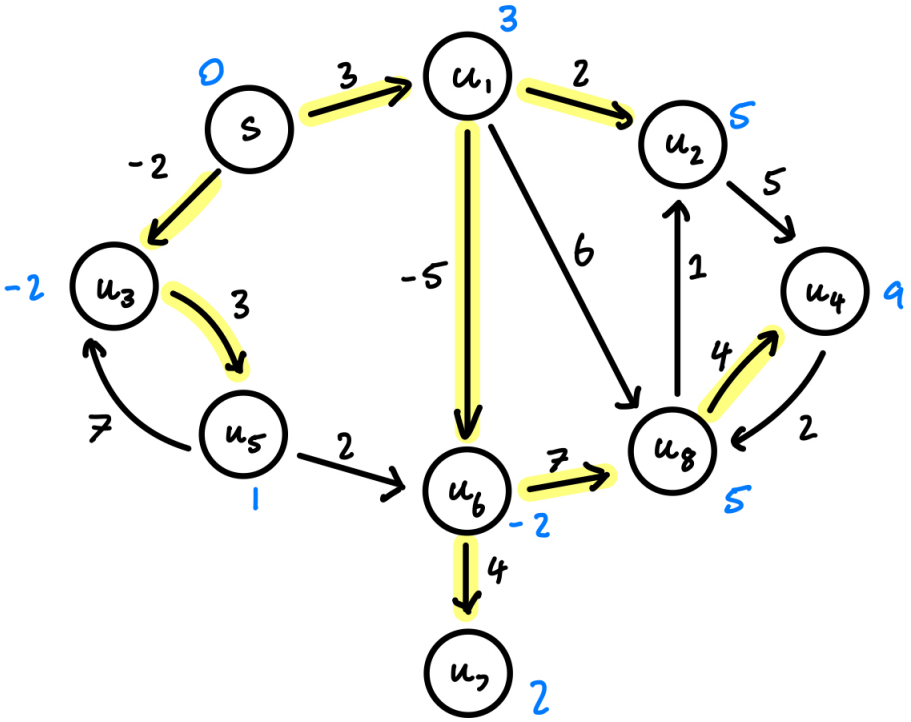
Run Bellman-Ford on the following graph using node s as the source. Below each node u, write the shortest path length from s to u. Mark the predecessor of u by highlighting it or making a bold arrow. You can assume that `graph.edges` produces the graph's edges in the following order:

$$(u_3, u_5), (u_1, u_2), (u_5, u_6), (u_4, u_8), (u_6, u_7), (s, u_1), (u_5, u_3),$$

$$(u_1, u_8), (u_6, u_8), (u_8, u_4), (s, u_3), (u_2, u_4), (u_1, u_6), (u_8, u_2)$$



Solution:



Problem 2.

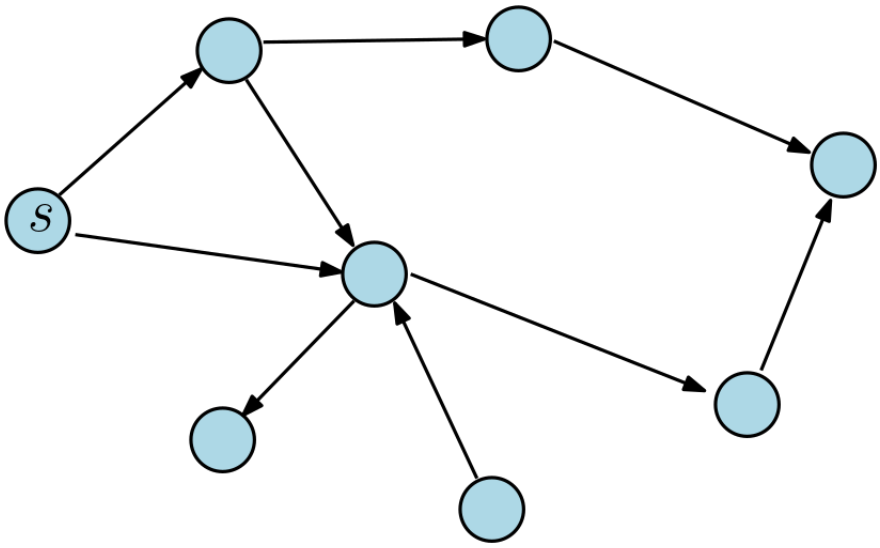
Recall that the Bellman-Ford algorithm (with early stopping) will terminate early if, after updating every edge, no predecessors have changed. Suppose it is known that the greatest shortest path distance in a graph $G = (V, E)$ has $\Theta(\sqrt{V})$ edges. What is the worst case time complexity of Bellman-Ford when run on this graph? State your answer using Θ notation.

Solution: The loop invariant for Bellman-Ford says that after updating all edges α times, all nodes whose shortest path distance is $\leq \alpha$ has the correct shortest path distance. Since the longest shortest path is $\Theta(\sqrt{V})$, we'll find all shortest paths after $\Theta(\sqrt{V})$ (outer) iterations. Every iteration involves $\Theta(E)$ work because it updates every edge, so the total work of the loop is $\Theta(E\sqrt{V})$.

We also spend $\Theta(V)$ time in the set up to initialize the estimated distance for each node in the graph. Therefore the total time is $\Theta(V + \sqrt{V}E)$.

Problem 3.

Suppose we are given a directed and weighted graph $G = (V, E)$ that look like the following figure. It is known that there are no negative cycles.



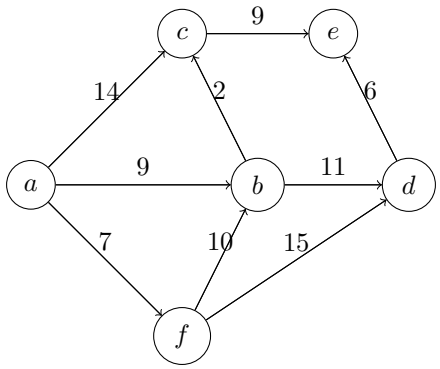
Suppose the Bellman-Ford algorithm **with early stopping** is run on this graph shown above using node s as the source (the edge weights are purposely not shown). In the worst case, how many iterations of the outer for-loop of Bellman-Ford will be performed? Briefly justify your answer.

Solution: The answer is 5.

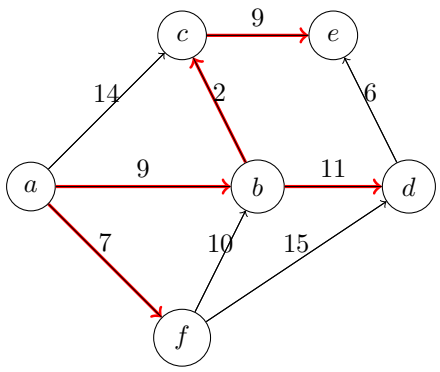
The longest simple directed path has length 4. Hence after $4 + 1$ iteration, the early-stopping criterion will be satisfied and the algorithm will terminate early.

Problem 4.

Run Dijkstra's Algorithm on the following graph using node a as the source. Below each node u, write the shortest path length from a to u. Mark the predecessor of u by highlighting it or making a bold arrow.



Solution:



est = { 'a': 0, 'b': 9, 'c': 11, 'd': 20. 'e': 20, 'f': 7 }

Problem 5.

True or False. Suppose $G = (V, E, \omega)$ is a weighted graph for which all edges are positive, except for those edges of a node s which may or may not be negative. If Dijkstra's algorithm is run on G with s as the source, the correct shortest paths will be found. Assume that the graph does not have any negative loops. Justify your answer.

Solution: True. The reason that Dijkstra's algorithm can fail in the presence of negative edges is that it cannot rule out that a negative edge that it has not seen will cause a path which is currently longer to eventually become shorter. But if all of the negative edges are attached to the source, the rest of the edges are still positive; and so there is no way in which adding edges to a path can decrease its length.

Problem 6.

Suppose $G = (V, E, \omega)$ is a weighted graph without negative cycles. Suppose we have an edge $(u, v) \in E$ whose weight is $w(u, v) = 5$. Let $s \in V$ be the source node. Let $\delta(s, u)$ and $\delta(s, v)$ denote the shortest path distance from s to u and to v, respectively.

a) (True or False): Now suppose that G is a **directed** graph. Then, it is possible that $\delta(s, u) = 50$ but $\delta(s, v) = 10$.

Solution: True. It is possibe that there is a directed path from s to v of length 10. For example, it is possible that we have the following directed graph $G = (V, E; w)$, where w is the weight function: $V = \{s, u, v\}$, $E = \{(s, u), (s, v)\}$, and the weights are $w(s, u) = 50$ and $w(s, v) = 10$. It is easy to see that in this case, $\delta(s, u) = 50$ and $\delta(s, v) = 10$.

b) (True or False): Now suppose that G is a **undirected** graph. Then, it is possible that $\delta(s, u) = 50$ but $\delta(s, v) = 10$.

Solution: False. In this case, note that we have edge $(u, v) = (v, u)$. So if we have $\delta(s, v) = 10$, and $w(u, v) = w(v, u) = 5$, then we know that the shortest path distance from s to u is at most the length of shortest path from s to v followed by edge (v, u); namely, $\delta(s, u) \leq \delta(s, v) + w(v, u) = 15$. So it is not possible that $\delta(s, u) = 50$.

Problem 7.

Consider the following scenario:

Lucas is hungry, and he wants to find a restaurant to eat at on campus. Unfortunately, all the restaurants on campus have a wait time.

Lucas has a weighted undirected graph $G = (V, E, \omega)$ of the campus, where notable locations are nodes, and the edge weights denotes time it takes to traverse that edge. Among the notable locations, some are restaurants. Each restaurant is marked with an associated wait time, indicating how long Lucas would need to wait for food *after arriving at the restaurant.*

Design an algorithm that **for all possible locations** of Lucas, $\forall v \in V$, computes the minimum time Lucas needs to spend before he can get food. Explain your algorithm in words and state its time complexity.

To receive full credit, your solutions must have optimal time complexity.

You may assume the graph is implemented using dictionary of sets.

Hint: Modify the graph or construct a new graph so that edge weights can include the information of wait times.

Solution: Note that this problem is essentially asking for an algorithm that computes the minimum (distance + wait time) to any restaurant for **all** starting nodes.

We would like to formulate the problem in a way that we can use the algorithms we have learned for finding shortest paths from a source node to all other nodes. To do so, we modify the graph as follows.

In order to include the wait times at restaurants as edges weights, we can add a new dummy node s to the graph. This node s has an edge to each of the restaurants. For any restaurant r we set the weight of the edge sr to be the wait time at restaurant r. Adding a node takes $\Theta(1)$ time, and connecting it with the restaurants takes $\Theta(R)$ time if there are R restaurants. Let us call this new graph G'.

A shortest path from s to a node $v \in V$ consists of the new edge from s to a restaurant r (which represents the wait time at r), plus a shortest path from the restaurant r to node v. The length of such shortest path is the same as the minimum time Lucas needs to spend before he can get food, starting at node v.

Hence, in order to solve the problem, we can run Dijkstra's algorithm on graph G' with s as the source node. This returns a shortest path from s to all the nodes in the graph. The time complexity required by this approach is $\Theta((V + E) \log V) + \Theta(R) = \Theta((V + E) \log V)$.