CSCI 432 Handout 12: Graph Algorithm Recap

Name: _____

March 31, 2026

Shortest Paths

Input: A (directed or undirected) weighted graph such that all edge weights are positive, $G = (V, E, \omega)$, where $\omega \colon E \to R_+$, a start node s, and an end node t.

Output: A shortest path between s and t.

1. Give an example of a graph such that the shortest path IS NOT unique..
   Answer

2. Give an example of a graph such that the shortest path IS unique..
   Answer

3. Give an example of a graph for which there is no path between s and t.
   Answer

Dijkstra's Algorithm

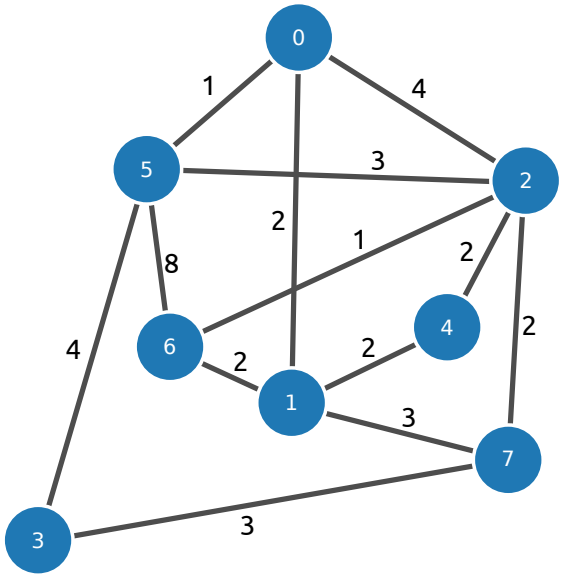
Let's remember how Dijkstra's algorithm works.

Initial set-up: We start create a priority queue where each vertex has priority ∞, except s, which has priority 0.

The main loop: We pop off the next element x of the priority queue, as we now know it's distance to the start s. We then perform the 'relaxation' step of updating all priorities to allow for going through x. If going through x is shorter than tentative distance ($priority(x) + w(x, v)$ ¡ current priority), then we update the priority.

When t is popped, we return the priority of t.

1. Walk through this algorithm for finding the shortest distance between 0 and 6 in the following graph:



2. What is the runtime of Dijsktra's algorithm? **Answer**

3. Write psuedo-code so that the algorithm returns a shortest path between s and t, not just the distance. **Answer**

4. True or false: If we don't stop when we pop t, then we wind up with a minimum spanning tree. **Answer**

5. What is the loop invariant for Dijkstra's algorithm (here, just think of the loop for the algorithm that returns the distance)? **Answer**

Other Well-Known Graph Algorithms

The following algorithms are well-known graph algorithms. For each algorithm, (a) recall or lookup the algorithm, maybe writing pseudocode and identifying the runtime; (b) work through a small example; (c) identify the loop invariant.

1. Bellman–Ford: This algorithm computes the shortest path from s to all other vertices. Unlike Dijkstra's algorithm, it allows negative edge weights (and can detect negative cycles). The initialization of Bellman–Ford is similar to that of Dijkstra (all vertices are labeled with ∞ as the estimated distances, except s is labeled with 0). Bellman–Ford does not use a priority queue though, it relaxes every edge every time ... stopping after $|V|$ iterations.
2. Floyd–Warshall algorithm: Another algorithm for computing shortest paths, allows negative weights (but no negative cycles). This algorithm uses dynamic programming to compute all pairs shortest paths. Along the way, this algorithm computes, for each pair $(v, w) \in V^2$ and each $k \in \{1, 2, \ldots, |V|\}$, the length of the shortest path from v to w only using the first k vertices.
3. Prim's Algorithm. Computes the MST of a weighted (undirected) graph. Similar to Dijkstra, it starts from an arbitrary node and uses a priority queue.