Exercise 5: Graphs

Due date: August 19, 11:59pm

Instructions:

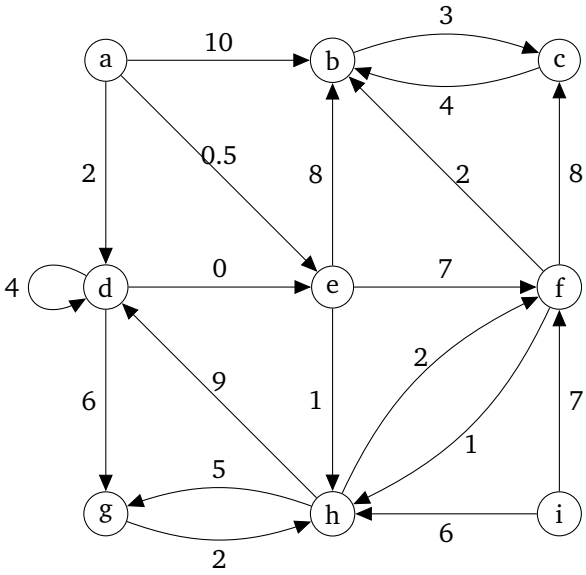
Submit a typed or neatly handwritten scan of your responses to the "Exercise 5" assignment on Gradescope here: https://www.gradescope.com/courses/52323. Make sure to log in to your Gradescope account using your UW email address to access our course.

For more details on how to submit, see
https://courses.cs.washington.edu/courses/cse401/18au/hw/submitting_hw_guide.pdf.

These problems are meant to be done **individually**. If you want to discuss these problems with a partner or group, make sure that you're writing your answers individually later on. Check our course's collaboration policy if you have questions.

1. Running Dijkstra's algorithm

Consider the following graph:



Run Dijkstra's algorithm on this graph, starting on node a. To do this, fill out the table below and be sure to show your work (cleanly cross out old values for distance and predecessor when updating existing values. For an example of this, see the Week 7 section slides example of Dijkstra's algorithm). If your handwriting is illegible, we may not be able to grade your subimssion so you might consider typing up the table as well.

When you choose a new vertex to process, choose the alphabetically earlier vertex in the case of a tie.

vertex	distance	predecessor	processed
a	0	None	
b	∞		
c	∞		
d	∞		
e	∞		
f	∞		
g	∞		
h	∞		
i	∞		

2. Using Dijkstra's algorithm

In this question, we will think about how to answer shortest path problems where we have more than just a single source and destination. Answer each of the following in English (not code or pseudocode). **Each subpart requires at most a few sentences to answer.** Answers significantly longer than required will not receive full credit.

You are in charge of routing ambulances to emergency calls. You have k ambulances in your fleet that are parked at different locations, and you need to dispatch them to an emergency to help as soon as possible. You have a map of Seattle represented as the adjacency list for a weighted, directed graph G with $|V|$ vertices and $|E|$ edges, where edge weights are positive numbers representing how long it takes to travel from the source to the destination. (The number of ambulances, k, is significantly less than the number of vertices $|V|$.) You also have a list of vertices representing the locations of each of your ambulances and the vertex representing where the new emergency is located. Figure 1 shows an example graph.

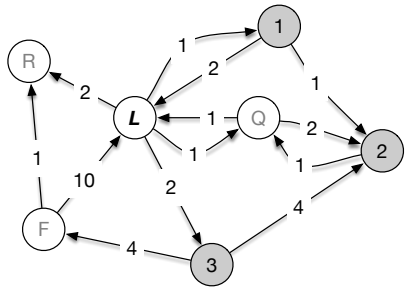


Figure 1: An example graph where $k = 3$ (the highlighted vertices). The emergency is L, and 1, 2, 3 are ambulances. F, R, Q are other intermediate locations. The shortest path from 1, 2, and 3 to L are 1-L, 2-Q-L, and 3-2-Q-L respectively.

(a) First, let's assume that you cannot alter the given graph and see what we can do.

(i) Describe how you would use Dijkstra's algorithm to output each of the shortest paths from each of the k ambulances to the emergency vertex. You should describe how to output the full routes, not just how long it takes. (Note: don't describe how Dijkstra's algorithm works by using a priority queue, updating the distances, etc. – describe the inputs and outputs and how those are used.)

(ii) What is the runtime for this process? Provide a simplified big-Θ for the worst-case runtime in terms of k, $|V|$, and/or $|E|$. Use the final version of Dijkstra's algorithm pseudocode from the lecture slides (Lecture 17) to form your answer.

(iii) How much extra space does this process require? Provide a simplified big-Θ for the worst-case memory usage in terms of k, $|V|$, and/or $|E|$. **Hint:** consider the space required for the table of information built in Dijkstra's algorithm ($\Theta(|V|)$ and the space required for the output.

(b) Now, by modifying the graph representation, we're going to discover a more efficient algorithm for this problem. Suppose we made a copy of the graph G' where every edge is reversed (also represented using an adjacency list). That is, every edge from u to v in the original graph has a corresponding edge from v' to u' with the same weight in the new graph (where v' and u' represent the copies of v and u respectively).

(i) What is the runtime of making this reversed copy of the graph? Provide a simplified big-Θ for the runtime in terms of k, $|V|$, and/or $|E|$, assuming that our adjacency list uses hash dictionaries where each `put` takes constant time.

(ii) How much extra space does it take to store the reversed copy of the graph? Provide a simplified big-Θ for the worst-case memory usage in terms of k, $|V|$, and/or $|E|$.

(iii) Describe how you would use Dijkstra's algorithm on the reversed graph to output the shortest paths from each ambulance to the emergency vertex. (Again, you should output the full route, not just how long it takes.)

(iv) What is the runtime for this entire process (including creating the reversed graph and running Dijkstra's algorithm)? Provide a simplified big-Θ for the worst-case runtime in terms of k, $|V|$, and/or $|E|$. As mentioned before, use the final version of Dijkstra's algorithm pseudocode from the lecture slides (Lecture 17) to form your answer.

(v) How much extra space does this entire process require (including creating the reversed graph and running Dijkstra's algorithm)? Provide a simplified big-Θ for the worst-case memory usage in terms of k, $|V|$, and/or $|E|$. (Reminder: the space complexity of Dijkstra's algorithm is $\Theta(|V|)$.

(vi) How is this algorithm better than the one in part (a)?

(vii) How is it worse than the one in part (a)?

(c) Now, assume we need to route only the closest ambulence to an emergency. With this change, we should be able to use an algorithm that does not require creating a copy of the entire graph.

(i) Add a "dummy node" (i.e., a new vertex that doesn't represent any real location) to the graph. We want to run Dijkstra's with our dummy node as the source. How should we connect it to the rest of the graph such that we can later run Dijkstra's only once to find the shortest path for the closest ambulance to follow in real life? Be sure to describe both the direction and weight of any edges you add.

(ii) What is the runtime for adding the dummy node and new edges? Provide a simplified big-Θ for the worst-case runtime in terms of k, $|V|$, and/or $|E|$, assuming that our adjacency list uses hash dictionaries where each `put` takes constant time.

(iii) How much extra space does it take to add the dummy node and new edges? Provide a simplified big-Θ for the worst-case memory usage in terms of k, $|V|$, and/or $|E|$.

(iv) In your modified graph, once you have run Dijkstra's algorithm, how do you tell which ambulance is closest to the emergency, and how do you recover the shortest path?

(v) What is the runtime for this entire process (including adding the dummy node and new edges and running Dijkstra's algorithm)? Provide a simplified big-Θ for the worst-case runtime in terms of k, $|V|$, and/or $|E|$. As mentioned before, use the final version of Dijkstra's algorithm pseudocode from the lecture slides (Lecture 17) to form your answer.

(vi) How much extra space does this entire process require (including adding the dummy node and new edges and running Dijkstra's algorithm)? Provide a simplified big-Θ for the worst-case memory usage in terms of k, $|V|$. (Reminder: the space complexity of Dijkstra's algorithm is $\Theta(|V|)$.

(vii) How is this algorithm better than the one in part (b)?

(viii) How is it worse than the one in part (b)?

3. Modeling with graphs

Suppose you are working for SpaceY, a private company planning exploration of Mars. You have just built a rover that will be sent to Mars. The rover has two modes.

- Bulldozer mode: in bulldozer mode, the rover will permanently clear any obstacle from its path, bulldozer mode uses an extremely large amount of power.
- Traveling mode: in traveling mode, the rover does not drain battery, but it cannot move around any obstacles.

In Bulldozer mode, the power used by the rover is proportional to the length of the route it travels.

You have a set of locations on Mars that the rover must visit, and the length of the direct route between each pair. Your goal is to plan which areas to bulldoze so you can reach every location in traveling mode, while using as little power as possible.

(a) Explain how you would model this scenario as a graph. Answer the following questions in bullet points with short sentences, then give an overall description on anything else that is relevant:

(i) What are your vertices and what information is stored in each vertex?

(ii) What are your edges and what information is stored in each edge?

(iii) Is this a weighted graph or an unweighted one? Why?

(iv) Is this a directed or undirected graph? Why?

(v) Do you permit self-loops (i.e. edges from a vertex to itself)? Parallel edges (i.e. more than one copy of an edge between the same location)? Why?

(vi) If there are any other relevant details about your model, describe them here:

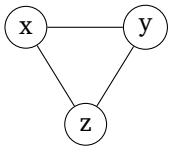
(vii) What graph algorithm do you run to find the routes to bulldoze?

(viii) Briefly explain how to convert the output of your algorithm to a plan of how to bulldoze and travel. Your answer should include which routes you are bulldozing, and how/when you will use traveling mode. (2-3 sentences)

(ix) Give a simplified big-Θ bound for the worst case running time of your algorithm from part (vii) in terms of k, the amount of locations on Mars.

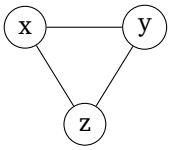
4. Shortest Path vs Minimum Spanning Tree

(a) The graph below has three vertices: x, y, z and three undirected edges. Label the edges with positive weights such that the shortest path produced by Dijkstra's algorithm from source vertex x to y is just the edge from x to y, and this edge is not included the MST of this graph.



(b) To verify your answer from above is correct, follow the instructions below.

(i) Copy the graph's edge weights above, and circle the shortest path edges on the graph below given by running Dijkstra's algorithm from source vertex x.



(ii) Copy the graph's edge weights above, and circle the edges in the minimum spanning tree given by Kruskal's algorithm on the graph below.