Homework 6: Analysis, sorting, and graphs

Due date: May 29, 11:59pm

Instructions:

Submit a typed or neatly handwritten scan of your responses to the "Homework 6" assignment on Gradescope here: https://www.gradescope.com/courses/47703. 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. Sorting algorithms

Answer each of the following in English (not code or pseudocode). **Each subpart requires at most 2-4 sentences to answer.** Answers significantly longer than required will not receive full credit.

(a) Recall that merge sort works by taking an array, splitting it into two pieces, recursively sorting both pieces, then combining both pieces in $\mathcal{O}(n)$ time. Suppose we split the array into *three* equally sized pieces instead of two. And after we finish recursively sorting each of the three pieces, we first merge two of the three pieces together, then we merge that with the third piece to get the final sorted result.

(i) Write the recurrence for this variation of merge sort

(ii) Use the master theorem to give an asymptotic bound for this variation of merge sort. Show your work.

(iii) Compare the recurrence and bound of this variation with that of the standard merge sort. Is this variation better or worse than the standard merge sort? Justify your answer.

(b) Suppose you are designing a search aggregator, which for a given query fetches search results from two different search engines and presents an intersection of the two search results. Here is a simplified version of this problem: Given two sorted integer arrays of lengths m and n, return a new array with elements that are present in both input arrays. The input array may contain duplicates, but there should be no duplicates in the output array. For example, if the input arrays are [17, 23, 23, 35, 43, 47, 69, 78, 80, 84, 84, 86] and [23, 35, 50], the output array should be [23, 35].

(i) Describe a brute force solution. What is the worst-case tight big-$\mathcal{O}$ time complexity of this brute solution?

(ii) Describe a solution that has worst-case tight-$\mathcal{O}(m \log n)$ time complexity.

(iii) Describe a solution that has worst-case tight-$\mathcal{O}(m + n)$ time complexity.

(iv) Between subparts (ii) and (iii) above, is one solution always better than the other (in terms of runtime)? If so, which one and why? If not, describe the situation (in terms of m and n) when one is better than the other.

(c) You're given an array where each element is an (age, name) pair representing guests at a DC-Marvel Universe party in Shire. You have been asked to print each guest at the party in ascending order of their ages, but if more than one guests have the same age, only the one who appears first in the array should be printed. For example, if the input array is

> [(23, Noah), (2000, Gandalf), (50, Frodo), (47, Emma), (23, Sophia), (83200, Superman), (23, Alice), (47, Madison), (47, Mike), (150, Dumbledore)]

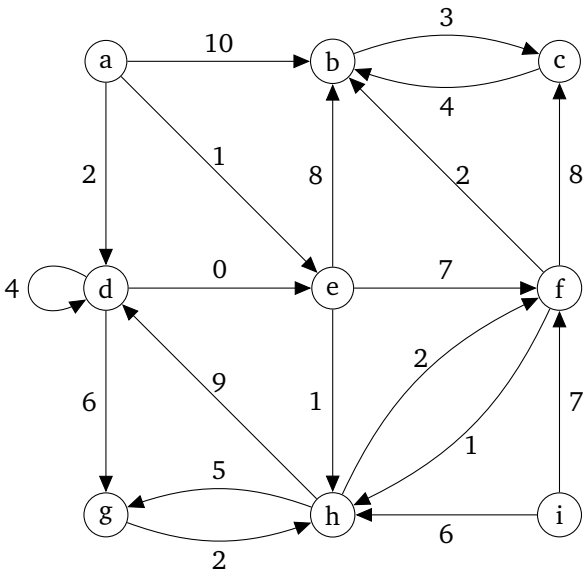
the output should be

> (23, Noah) (47, Emma) (50, Frodo) (150, Dumbledore) (2000, Gandalf) (83200, Superman)

Describe a solution that takes $\mathcal{O}(1)$ space in addition to the provided input array. Your algorithm may modify the input array. This party has some very old guests and your solution should still work correctly for parties that have even older guests, so your algorithm can't assume the maximum age of the partygoers. Give the worst-case tight-$\mathcal{O}$ time complexity of your solution.

2. 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 8 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.

In the case of a tie, add the vertex that comes first alphabetically.

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

3. 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.

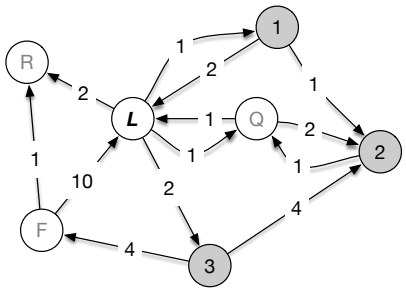


Figure 1: An example graph where $k = 3$ (the highlighted vertices). The emergency is L, and 1, 2, 3 are ambulances. F, R, Q are other intermediate locations. The shortest path from 1, 2, and 3 to L are 1-L, 2-Q-L, and 3-2-Q-L respectively.

(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 ambulance to the emergency vertex. (You should output the full route, not just how long it takes.)

(ii) What is the runtime for this process? Provide a simplified, tight big-$\mathcal{O}$ for the runtime in terms of k, $|V|$, and/or $|E|$. Use the final version of Dijkstra's algorithm pseudocode from the lecture slides (Lecture 22) to form your answer.

(iii) How much extra space does this process require? Provide a simplified, tight big-$\mathcal{O}$ for the 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 (tight-$\mathcal{O}(|V|)$) 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, tight big-$\mathcal{O}$ 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, tight big-$\mathcal{O}$ for the 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, tight big-$\mathcal{O}$ for the 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 22) 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, tight big-$\mathcal{O}$ for the memory usage in terms of k, $|V|$, and/or $|E|$. (Reminder: the space complexity of Dijkstra's algorithm is tight-$\mathcal{O}(|V|)$.)

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

(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, tight big-$\mathcal{O}$ 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.

(iii) How much extra space does it take to add the dummy node and new edges? Provide a simplified, tight big-$\mathcal{O}$ for the 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, tight big-$\mathcal{O}$ for the 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 22) 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, tight big-$\mathcal{O}$ for the memory usage in terms of k, $|V|$. (Reminder: the space complexity of Dijkstra's algorithm is tight-$\mathcal{O}(|V|)$.)

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

4. Modeling with graphs

Suppose we are trying to implement a program that finds a shortest path between any two locations within Seattle.

In case of a tie, the program should find us the route with the *fewest vertices on the path*. For example, suppose we are considering two different routes from UW to the Space Needle. Both routes are 3 miles long, but route one has 3 vertices and route two has five vertices. In this case, our algorithm should pick route one.

(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 edges?

(ii) What information do you store for each vertex and edge?

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

(iv) Is this a directed or undirected graph?

(v) Do you permit self-loops? Parallel edges?

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

(b) At a high-level, how you would modify Dijkstra's algorithm to find us the best route according to the specifications listed above. In particular, be sure to explain:

(i) How you combine or use different factors like road length and number of vertices in the path while finding the best route.

(ii) How you would modify Dijkstra's algorithm to break ties in the manner described above. (State which lines you would modify, if any, and/or lines you would add to the following pseudocode; state how and when you would check and resolve ties, e.g., during the BFS search or after the BFS search)

Use the pseudocode for Dijkstra's algorithm on the **next page** as a base for your response to this question; it will be helpful to list specific lines that you will be modifying to fit this specific problem. Be sure your algorithm makes sense when combined with the graph representation you chose in part a.

Answer in English (not code or pseudocode) and in at most a few sentences.

Answers significantly longer than required will not receive full credit.

```
Dijkstra(Graph G, Vertex source)
    // Set up.
    for (Vertex v : G.getVertices())
        v.dist = INFINITY
    G.getVertex(source).dist = 0
    initialize MPQ as a Min Priority Queue, add source

    // Main loop.
    while(MPQ is not empty)
        u = MPQ.removeMin()
        for (Edge (u, v) : u.getEdges(u)
            oldDist = v.dist
            newDist = u.dist+weight(u,v)

            if (newDist < oldDist)
                v.dist = newDist
                v.predecessor = u

                if (oldDist == INFINITY)
                    MPQ.insert(v, newDist)
                else
                    MPQ.updatePriority(v, newDist)
```