Homework 06: Analysis, sorting, and graphs

Due date: November 30, 2018 at noon.

Instructions:

Submit a typed or neatly handwritten scan of your responses on Canvas in PDF format.

Note: You will need to submit a separate PDF per each part, so you need to submit six PDFs. Each PDF should contain answers to questions from only one part.

Part 1. (15 points) The tree method

Submit your answers here: https://canvas.uw.edu/courses/1219673/assignments/4497460

Consider the following recurrence:

$$A(n) = \begin{cases} 5 & \text{if } n = 1 \\ 4A(n/2) + n^2 \log_2 n & \text{otherwise} \end{cases}$$

We want to find an *exact* closed form of this equation by using the tree method.

(a) (2 points) Draw your recurrence tree (as shown in the class; see lecture 18, slide 19).

(b) (1 point) What is the size of the input at level i (as in class, call the root level 0)?

(c) (2 points) what is the number of nodes at level i?

Note: let $i = 0$ indicate the level corresponding to the root node. So, when $i = 0$, your expression should be equal to 1.

(d) (3 points) What is the total work at the i-th *recursive* level? Be sure to show your work for full credit.

(e) (1 point) What is the last level of the tree?

(f) (2 points) What is the work done in the base case?

(g) (3 points) Combine your answers from previous parts to get an expression for the total work. Simplify the expression to a closed form. Be sure to show your work for full credit.

(h) (1 point) From the closed form you computed above, give a tight-$\mathcal{O}$ bound. No need to prove it. (Hint: You may need to simplify the closed form.)

Part 2. (13 points) Sorting algorithms

Submit your answers here: https://canvas.uw.edu/courses/1219673/assignments/4497462

Answer each of the parts 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) (5 points) 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.

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) (5 points) 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 big-$\mathcal{O}$ time complexity of this brute solution?

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

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

iv. Between subparts ii and iii above, is one solution always better than the other (in terms of runtime)? If yes, which one and why? If no, describe the situation (in terms of m and n) when one is better than the other.

(c) (3 points) 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 time complexity of your solution.

(d) (3 points (**Extra credit**)) Consider the same problem in part c, but this time there are only mortal humans in the party, and the oldest guest in the party is k years old. Give a solution that runs in $\mathcal{O}(n)$ time, where n is the number of party guests. You may use $\mathcal{O}(k)$ of additional memory.

Part 3. (10 points) Running Dijkstra's algorithm

Submit your answers here: https://canvas.uw.edu/courses/1219673/assignments/4497463

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 parts in English (not code or pseudocode). **Each bullet point requires at most a few sentences to answer.** Answers significantly longer than required will not receive full credit.

(a) (6 points) You and $k-1$ of your friends have all decided to meet at Odegaard for a study session. Unfortunately you're scattered all over campus, and you'd like to start studying as soon as possible.

You have a map of campus represented as the adjacency list for a weighted, directed graph (the graph has only positive edge weights). You have a list of the vertices representing the current locations of you and your friends. You also know which vertex corresponds to Odegaard. Figure 1 shows an example graph.

i. In this part, assume that you cannot alter the graph you have been given.

  i. Describe how to find the quickest way for each of you and your friends to arrive at Odegaard. You should say which algorithm(s) you run and any inputs you would pass into the functions described in class.

  ii. What is the big-$\mathcal{O}$ running time for this process in terms of k, $|V|$ and $|E|$?

iii. In this part, we will alter the graph to get a more efficient algorithm.

  i. Describe a modification you could make to the graph to make finding all of these paths more efficient. You do not need to provide code or pseudocode to describe your modification, but your modification must be something that could be implemented in $\mathcal{O}(|V| + |E|)$ time.

  ii. Once you make the modification, how do you find all the paths?

  iii. What is the big-$\mathcal{O}$ running time to find all of the paths? Your answer in this part should be more efficient than the previous algorithm.

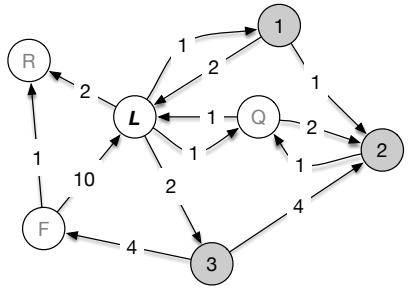


Figure 1: An example graph where $k = 3$ (the highlighted vertices). Odegaard library is L, and Q, F, R are other locations on campus. The shortest path from 1, 2, and 3 to L are 1-L, 2-Q-L, and 3-2-Q-L respectively.

(b) (4 points) You are in charge of routing ambulances to emergency calls. You have three ambulances in your fleet that are parked at different locations. There is an emergency and you have to dispatch the ambulance closest to the emergency to help as soon as possible.

You have a map of Seattle represented as the adjacency list for a weighted, directed graph (the graph has only positive edge weights). 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.

You could run Dijkstra's from each of the three ambulance vertices, but since every second counts, you decide you'd like an algorithm that has better constant factors (even if you can't improve the big-$\mathcal{O}$).

i. Describe a modification you can make to the graph (without changing any existing edges) so you only need to run Dijkstra's algorithm once. It should be possible to make your modification in constant time.

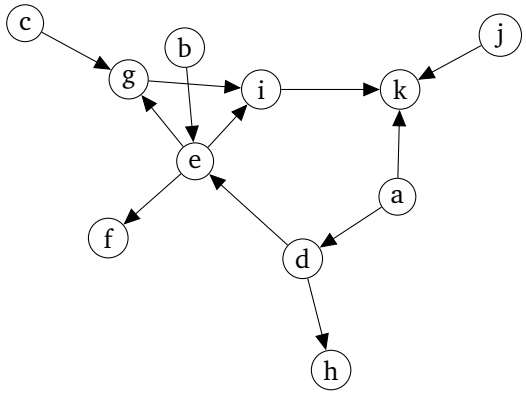
ii. In your modified graph, how do you tell which ambulance to send?

iii. Show an example graph before and after the modification.

Part 4. (10 points) Running topological sort

Submit your answers here: https://canvas.uw.edu/courses/1219673/assignments/4497464

Consider the following graph:



(a) (6 points) Suppose we run the topological sort algorithm on this graph. List three different possible valid outputs.

(b) (4 points) Suppose we want to modify this graph so that running topological sort is impossible – so that there is no valid output when we try running the algorithm.

We are allowed to change the graph by either adding or removing exactly one edge.

What edge would you add or remove? Briefly explain why your change causes the graph to no longer have a valid topological ordering.

Part 5. (10 points) Modifying Dijkstra's algorithm

Submit your answers here: https://canvas.uw.edu/courses/1219673/assignments/4497465

(a) (4 points) Suppose you are given a graph that has negative-cost edges, but no negative-cost cycles. You want to find the shortest path between two points and decide to do so using the following algorithm:

i. Add every edge by some constant k such that there are no more negative-cost edges.

ii. Run Dijkstra's algorithm to find the shortest path.

Unfortunately, this algorithm does not work. Give an example of a graph where this algorithm fails to return the correct answer. Be sure to explain why the algorithm fails on this graph.

Note: your example does not need to be complicated – you should need to use at most three or four nodes.

(b) (3 points) Suppose we have a version of Dijkstra's algorithm implemented using binary heaps with a $\mathcal{O}(\log(n))$ `decreaseKey` method. As we discussed in lecture, this version of Dijkstra's algorithm will have a worst-case runtime of $\mathcal{O}(|V|\log(|V|) + |E|\log(|V|))$.

Now, suppose we know we will always run this algorithm on a *simple* graph. Given this information, rewrite the worst-case runtime of Dijkstra's algorithm so it only uses the variable $|V|$. Briefly justify your answer.

(c) (3 points) Why does your answer to part (b) apply only to simple graphs? Why is it an inaccurate worst-case bound for non-simple graphs?

Part 6. (10 points) Modeling with graphs

Submit your answers here: https://canvas.uw.edu/courses/1219673/assignments/4497466

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) (7 points) 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?

(b) (3 points) 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 following pseudocode for Dijkstra's algorithm 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.

```
1:  function Dijkstra(Graph G, Vertex source)
2:      initialize distances to ∞
3:      mark source as distance 0
4:      mark all vertices unprocessed
5:      while there are unprocessed vertices do
6:          let u be the closest unprocessed vertex
7:          for each edge (u, v) leaving u do
8:              if u.dist + w(u, v) < v.dist then
9:                  v.dist = u.dist + w(u, v)
10:                 v.predecessor = u
11:         mark u as processed
```