Homework 02: Analysis, sorting, and graphs

Due date: March 2, 2018 at 11:30 pm

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 section.

1. The tree method and the master method

Submit your answers here: `https://canvas.uw.edu/courses/1124150/assignments/4107400`

Consider the following recurrence:

$$A(t) = \begin{cases} 5 & \text{if } t = 1 \\ 5A(t/3) + t^2 & \text{otherwise} \end{cases}$$

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

(a) Suppose we draw out the total work done by this method as a tree, as discussed in lecture. Including the base case, how many levels in total are there?

Your answer should be a mathematical expression which uses the variable T, which represents the original number we passed into $A(t)$.

(b) How many nodes are there on any given level i? Your answer should be a mathematical expression that uses the variable i.

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

(c) How much work is done on the i-th *recursive* level? Your answer should be a mathematical expression that uses the variables i and T.

(d) How much work is done on the final, *non-recursive* level? Your answer should be a mathematical expression that uses the variable T.

(e) What is the closed form of this recurrence? Be sure to show your work.

Note: you do not need to simplify your answer, once you found the closed form. Hint: You should use the finite geometric series identity somewhere while finding a closed form.

(f) Use the master theorem to find a big-Θ bound of $A(t)$.

2. Sorting algorithms

Submit your answers here: `https://canvas.uw.edu/courses/1124150/assignments/4107401`

(a) Suppose we have a version of quick sort which selects pivots using the median-of-three strategy and creates partitions in the manner described in lecture.

Explain how you would build an input array that would cause this version of quick sort to run in $\mathcal{O}\left(n^2\right)$ time.

Your answer should explain on a high level what your array would look like and what happens when you try running quick sort on it. You do not need to give a specific example of such a array, though you may if you think it will help make your explanation more clear.

(b) Recall that merge sort works by taking an array, splitting it into two pieces, recursively sorting both pieces, then combining both pieces in $\mathcal{O}\left(n\right)$ time.

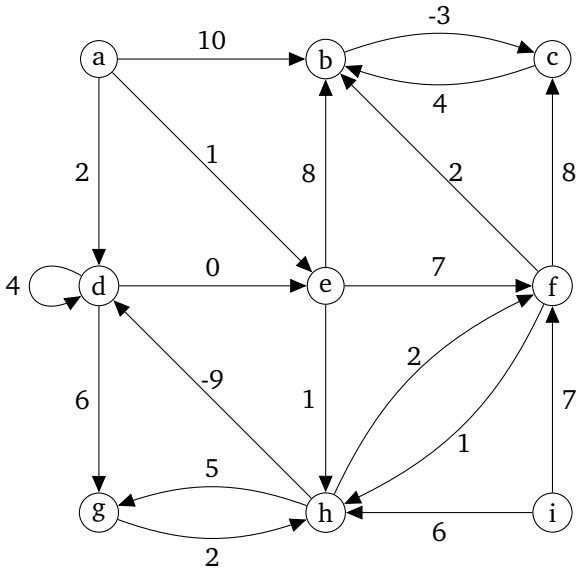
Suppose that instead we split the array into *three* equally sized pieces instead of two. After we finish recursing on each of the three pieces, we merge two of the three pieces together, then merge that with the third piece to get the final sorted result.

What is the worst-case asymptotic runtime of this variation on merge sort? Is it better or worse then the original version of merge sort? Be sure to justify your answers.

3. Running Dijkstra's algorithm

Submit your answers here: `https://canvas.uw.edu/courses/1124150/assignments/4107402`

Consider the following graph:



(a) Run Dijkstra's algorithm on this graph, starting on node a. Show the final costs of each vertex, as well as the edges that are selected by Dijkstra's algorithm.

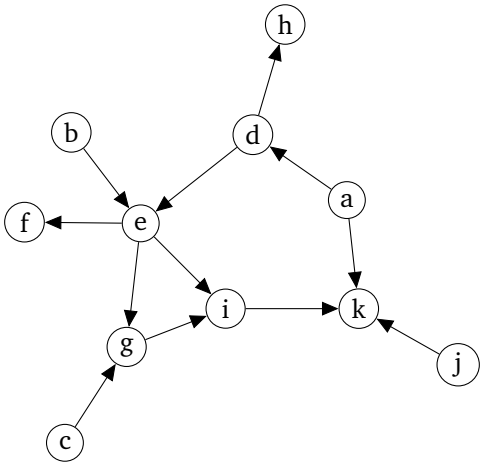
In the case of a tie, add the vertex that comes first alphabetically.

(b) Did Dijkstra's algorithm return the correct result? Explain why or why not.

4. Running topological sort

Submit your answers here: `https://canvas.uw.edu/courses/1124150/assignments/4107403`

Consider the following graph:



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

(b) 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.

5. Modifying Dijkstra's algorithm

Submit your answers here: `https://canvas.uw.edu/courses/1124150/assignments/4107404`

(a) 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:

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

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

6. Modeling with graphs

Submit your answers here: `https://canvas.uw.edu/courses/1124150/assignments/4107405`

Suppose we are trying to implement a program that finds a path between any two locations within Seattle that takes the *shortest amount of time* possible. Note that the time needed to travel down a particular stretch of road depends on several different factors such as the length of the road, the speed limit, and the amount of traffic.

In case of a tie, the program should find us the route with the *shortest physical distance*. For example, suppose we are considering two different routes from UW to the Space Needle. Based on current traffic conditions, both routes will take 15 minutes. However, route one is 3.5 miles long and route two is 3.7 miles long. In that case, our algorithm should pick route one.

(a) Explain how you would model this scenario as a graph. In particular, be sure to answer the following questions:

- What are your vertices and edges?
- What information do you store for each vertex and edge?
- Is this a weighted graph or an unweighted one?
- Is this a directed or undirected graph?
- Do you permit self-loops? Parallel edges?

(b) Explain 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:

- How you combine or use different factors like road length, the speed limit, or the amount of traffic while finding the best route.
- How you would modify Dijkstra's algorithm to break ties in the manner described above.

(c) Suppose we want to modify our program so that the user and pick whether they want to take a car or walk to the given destination.

This changes what paths the user may take. For example, if the user is taking a car, they can travel along freeways but not through parks or alleyways. Conversely, if the user is walking, they can travel through parks and alleyways but not freeways.

Explain how you would implement this new feature. Do you need to make any changes to your model? Do you need to modify how you implement Dijkstra's algorithm?

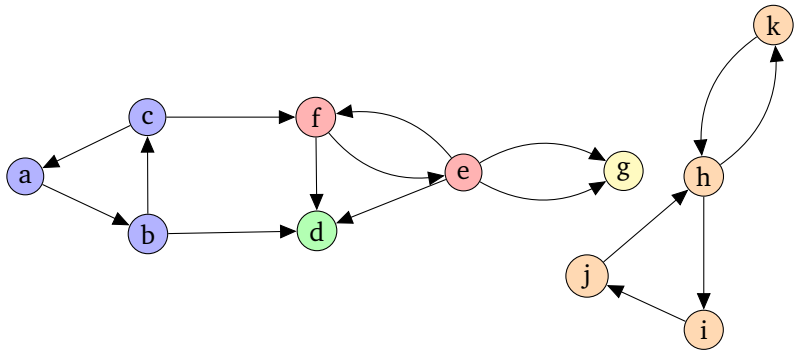
Extra credit: Strongly-connected components

NOTE: this problem is extra credit, and is optional.

Submit your answers here: `https://canvas.uw.edu/courses/1124150/assignments/4107406`

Given a directed graph, a *strongly connected component* (SSC) is any subgraph where there exists a path connected any vertex in that component to any other vertex in that component.

For example, consider the following graph:



There are several SSCs contained within this graph. For example, take the vertices a, b, and c. These vertices form a single SSC: there exists a path from any one of these vertices to another one.

This graph contains four more SSCs: the vertices f, e; the vertex d; the vertex g; and finally the vertices h, i, j, k. These SSCs are highlighted in the graph up above.

Your goal is to implement an algorithm named `getSCCs(graph)` that accepts some directed graph as input and returns all of the SCCs contained within that graph.

For example, if we received the above graph as input, we would return something like the following:

```
[[a, b, c], [d], [e, f], [g], [h, i, j, k]]
```

(a) Write an **English description** or **high-level pseudocode** describing an algorithm to perform this task. Note: do **NOT** submit Java code. We want to see a high-level description of the algorithm, not a low-level one.

Some notes:

- You do not need to worry about checking for or handling invalid inputs.
- Your algorithm does not need to be particularly efficient to get extra credit – we care more about the quality of your explanations and your analysis.
- If you want a challenge/get a few bonus points, try finding an algorithm that can complete this task in $\mathcal{O}(|V| + |E|)$ time.

(b) Justify the correctness of your algorithm. (Your goal here is to write an explanation that would convince a skeptical reader that your algorithm actually does work.)

(c) What is the worst-case asymptotic runtime of your algorithm in terms of $|V|$ and $|E|$? Be sure to justify your answer.