CS 161 **Problem Set 5**
Spring 2023 **Due:** Wednesday, May 17 at 1pm on Gradescope.

---

**Style guide and expectations:** Please see the "Homework" part of the "Resources" section on the webpage for guidance on what we are looking for in homework solutions. We will grade according to these standards.

Make sure to look at the "**We are expecting**" blocks below each problem to see what we will be grading for in each problem!

**Collaboration policy:** You may do the HW in groups of size up to three. Please submit one HW for your whole group on Gradescope. (Note that there is an option to submit as a group). See the "Policies" section of the course website for more on the collaboration policy.

Please list all members of your homework group and all other students in the class you have collaborated with on this homework, in accordance with the Collaboration Policy, at the beginning of each homework submission.

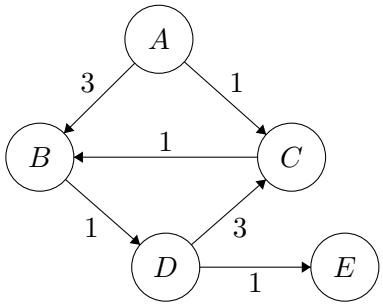
---

## Exercises

We recommend you do the exercises on your own before collaborating with your group.

---

1. **(6 pt.)** Consider the following directed graph $G$:




For the following parts you might want to use the website http://madebyevan.com/fsm/, which allows you to draw directed graphs in LaTeX. (Note: On a Mac, fn+Delete will delete nodes or edges). It is also fine to include an image created in your favorite drawing program, or a photo/scan of a hand-drawn graph.

(a) **(2 pt.)** Draw the DFS tree for $G$, starting from node $A$. Assume that DFS traverses nodes in alphabetical order. (That is, if it could go to either $B$ or $C$, it will always choose $B$ first).

[**We are expecting:** *A picture of your tree. No further explanation is required.*]

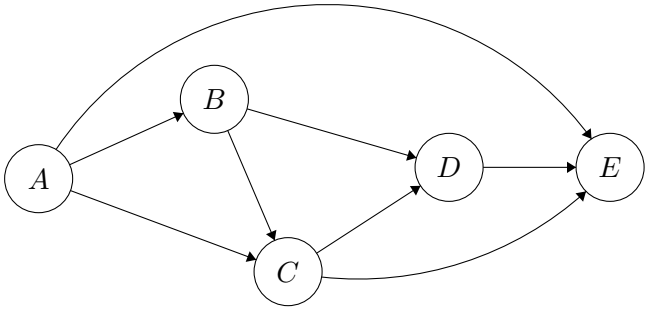
(b) **(2 pt.)** Draw the BFS tree for $G$, starting from node $A$. Assume that BFS traverses nodes in alphabetical order.

[**We are expecting:** *A picture of your tree. No further explanation is required.*]

(c) **(2 pt.)** Draw the "Dijkstra's algorithm tree" for $G$, starting from node $A$. Assume that Dijkstra's algorithm breaks ties in alphabetical order.

[**We are expecting:** *A picture of your tree. No further explanation is required.*]

2. **(4 pt.)** Consider the following directed acyclic graph (DAG):




In class, we saw how to use DFS to find a topological ordering of the the vertices; in the graph above, the unique topological ordering is $A, B, C, D, E$. We saw an example where we happened to start DFS from the first vertex in the topological order. In this exercise we'll see what happens when we start at a different vertex. Recall that when you run DFS, if it has reached everything it can but hasn't yet explored the graph, it will start again at an unexplored vertex.

(a) Run DFS starting at vertex $C$, breaking any ties by alphabetical order.[1]

i. What do you get when you order the vertices by **ascending** start time?

ii. What do you get when you order the vertices by **descending** finish time?

(b) Now run DFS starting at vertex $C$, breaking any ties by **reverse** alphabetical order.[2]

i. What do you get when you order the vertices by **ascending** start time?

ii. What do you get when you order the vertices by **descending** finish time?

[**We are expecting:** *For all four questions, an ordering of vertices. No justification is required.*]

[1] For example, if DFS has a choice between $B$ or $C$, it will always choose $B$. This includes when DFS is starting a new tree in the DFS forest.

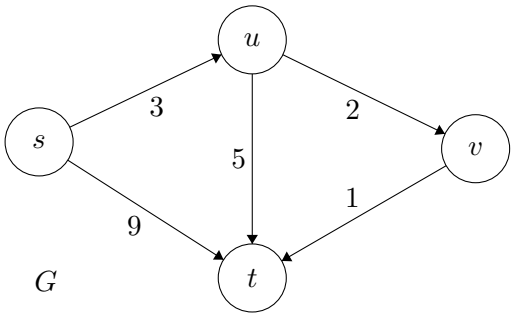
[2] For example, when DFS has a choice between $B$ or $C$, it will always choose $C$. This includes when DFS is starting a new tree in the DFS forest.

3. **(6 pt.)** In class, we saw pseudocode for Dijkstra's algorithm which returned shortest distances but not shortest paths. In this exercise we'll see how to adapt it to return shortest paths. One way to do that is shown in the pseudocode below:

```
Dijkstra_st_path(G, s, t):
    for all v in V, set d[v] = Infinity
    for all v in V, set p[v] = None
    // we will use the information p[v] to reconstruct the path at the end.
    d[s] = 0
    F = V
    D = []  // D is the list of "done" vertices
    while F isn't empty:
        x = a vertex v in F so that d[v] is minimal
        for y in x.outgoing_neighbors:
            d[y] = min( d[y], d[x] + weight(x,y) )
            if d[y] was changed in the previous line, set p[y] = x
        F.remove(x)
        D.add(x)

    // use the information in p to reconstruct the shortest path:
    path = [t]
    current = t
    while current != s:
        current = p[current]
        add current to the front of the path
    return path, d[t]
```

Step through `Dijkstra_st_path`$(G, s, t)$ on the graph $G$ shown below. Complete the table below (on the next page) to show what the arrays `d` and `p` are at each step of the algorithm, and indicate what path is returned and what its cost is. If it is helpful, the LaTeXcode for the table is in the LaTeX template.




[**We are expecting:** *The following things:*

- *The table below filled out*

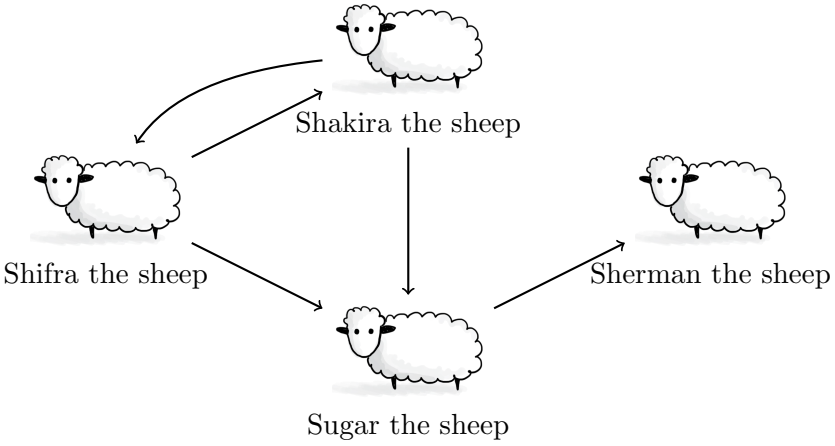
- *The shortest path and the cost that the algorithm returns.*

*No justification is required.*]

| | $\mathtt{d}[s]$ | $\mathtt{d}[u]$ | $\mathtt{d}[v]$ | $\mathtt{d}[t]$ | $\mathtt{p}[s]$ | $\mathtt{p}[u]$ | $\mathtt{p}[v]$ | $\mathtt{p}[t]$ |
|---|---|---|---|---|---|---|---|---|
| When entering the first while loop for the first time, the state is: | 0 | $\infty$ | $\infty$ | $\infty$ | None | None | None | None |
| Immediately after the first element of $D$ is added, the state is: | 0 | 3 | $\infty$ | 9 | None | s | None | s |
| Immediately after the second element of $D$ is added, the state is: | | | | | | | | |
| Immediately after the third element of $D$ is added, the state is: | | | | | | | | |
| Immediately after the fourth element of $D$ is added, the state is: | | | | | | | | |

## Problems

4. **(8 pt.)** (**Wake up, Sheeple!**) You arrive on an island with $n$ sheep. The sheep have developed a pretty sophisticated society, and have a social media platform called Baaaahtter (it's like Twitter but for sheep[3]). Some sheep follow other sheep on this platform. Being sheep, they believe and repeat anything that they hear. That is, they will re-post anything that any sheep they are following said. We can represent this by a graph, where $(a) \to (b)$ means that $(b)$ will re-post anything that $(a)$ posted. For example, if the social dynamics on the island were:




then Sherman the Sheep follows Sugar the Sheep, and Sugar follows both Shakira and Shifra, and so on. This means that Sherman will re-post anything that Sugar posts, Sugar will re-post anything by Shifra and Shikira, and so on. (If there is a cycle then each sheep will only re-post a post once).

For the parts below, let $G$ denote a directed, unweighted graph on $n$ sheep. Let $m$ denote the number of edges in $G$.

(a) **(2 pt.)** Call a sheep an **influencer** if anything that they post eventually gets re-posted by every other sheep on the island. In the example above, both Shifra and Shakira are influencers.

Prove that all influencers are in the same strongly connected component of $G$, and every sheep in that component is an influencer.

[**We are expecting:** *A short but rigorous proof.*]

(b) **(4 pt.)** Suppose that there is at least one influencer. Give an algorithm that runs in time $O(n+m)$ and finds an influencer. You may use any algorithm we have seen in class as a subroutine.

[**We are expecting:** *The following things:*

[3] Also my new start-up idea

- *Pseudocode or a very clear English description of your algorithm*
- *an informal justification that your algorithm is correct*
- *an informal justification that the running time is $O(n+m)$*

*You may use any statement we have proved in class without re-proving it.*]

(c) **(2 pt.)** Now suppose that you don't know whether or not there is an influencer. Give an algorithm that runs in time $O(n+m)$ and either returns an influencer if there is one, or else returns "`no influencer.`" You may use any algorithm we have seen from class as a subroutine, and you may also use your algorithm from part (b) as a subroutine.

[**We are expecting:** *The following things:*

- *Pseudocode or a very clear English description of your algorithm*
- *an informal justification that your algorithm is correct*
- *an informal justification that the running time is $O(n+m)$*

*You may use any statement we have proved in class without re-proving it.*]

5. **(6 pt.)** (**Dijkstra with negative edge weights?**) For both of the questions below, suppose that $G$ is a connected, directed, weighted graph, which may have negative edge weights, containing vertices $s$ and $t$, and refer to the pseudocode for `Dijkstra_st_path` from Exercise 3. Also suppose that there *is* some path from $s$ to $t$ in $G$.

   (a) **(2 pt.)** Give an example of a graph where there is a path from $s$ to $t$, but no shortest path from $s$ to $t$. (Note that in a directed graph, a *path* must follow the direction of the edges; recall that a *shortest path* is one which minimizes the sum of the edge weights along that path).

   [**We are expecting:**

   - *A small example (graph with at most* 5 *vertices)*
   - *An explanation of why there is no shortest path from $s$ to $t$.*

   ]

   (b) **(4 pt.)** Give an example of a graph where there *is* a shortest path from $s$ to $t$, but `Dijkstra_st_path`$(G, s, t)$ does not return one.

   [**We are expecting:**

   - *A small example (graph with at most* 5 *vertices)*
   - *An explanation of what* `Dijkstra_st_path` *does on this graph and why it does not return a shortest path. Note that even if* `Dijkstra_st_path` *returns the wrong distance from $s$ to $t$, it might still return the correct path!*

   ]

6. **(6 pt.)** (**The EthiCS of Baaaahtter and shortest paths**)

(a) **(3 pt.)** In problem 4, the sheep on the island will re-post anything that any sheep they follow said on Baaaahtter. In real life, most users will only comment or repost a small subset of the posts they see, so the kind of engagement we see in Baaaahtter is something social media companies need to deliberately optimize for. However, as seen with Facebook and Twitter a few years ago, optimizing for engagement can have adverse consequences, including proliferating polarizing opinions and false information. What other values are being set aside when we optimize for high engagement? If you were working on the recommendation algorithm at a social media company and you saw the potential for these problems, what are some steps you or the company could take in developing the recommendation algorithm to mitigate them? Here are some examples for inspiration.

[**We are expecting:** *3-5 sentences explaining (1) 2-3 other values that we overlook when prioritizing high engagement, and (2) at least one way to change a recommendation algorithm to address issues created by prioritizing high engagement*]

(b) **(1 pt.)** You've learned several algorithms for finding the shortest path in a graph. Urban planners often utilize these algorithms when planning walkable cities (see a quick overview of the 15-minute city concept or its original proposal). Imagine you are tasked with the role of making a city more walkable by modifying the existing zoning laws to allow for some commercial use units to open closer to existing residential buildings. To do this, you make an algorithm that identifies locations for commercial units that minimize the distance to a set number of existing residential units (concretely, your algorithm minimizes the average distance between residential units and their closest commercial location). Then you sell these locations to businesses that would diversify the types of amenities available in the neighborhood. However, you realize that this one way of optimizing distance to residential buildings can overlook other aspects of creating diverse and accessible neighborhoods. What are other considerations we overlook when only optimizing for distance traveled? Here are a few articles for some inspiration: Bloomberg, WEF.

[**We are expecting:** *1-2 sentences explaining a consideration we overlook when we only consider the distance traveled.*]

(c) **(2 pt.)** When we work with algorithms, we often need to change or ignore aspects of a real-world situation in order to turn it into a solvable problem. For example, in order to write an algorithm that sorts a list of amusement parks, we need to choose a quantity for comparison (total number of rides, total number of burritos, etc.).

- **Abstraction** is when we omit details of the real-world situation. For example, we could omit the kind of thing being sorted by our algorithm, what condition it is in, or how long it has been in the list.
- **Idealization** is when we deliberately change aspects of the real-world situation. For example, we could round specific numbers and change them into whole numbers.

Compare the problem in part (b) of finding a location that minimizes the distance to existing residences to the real-world problem of designing walkable human-centric neighborhoods. Identify one abstraction and one idealization.

[**We are expecting:** *2-3 sentences which identify one aspect of the problem setup above as an abstraction and one aspect as an idealization of the real-world problem. For each, explain why you believe it is an idealization or abstraction.*]

7. **[Please Fill Out This Survey] (1 pt.)**

   In collaboration with Professor Shima Salehi of the Stanford Ed School and her team, we have created a short google form on your experience with the class's problem solving guide. SUNet IDs will be collected, but we will only use hashes of IDs for the study, so the responses will be effectively anonymous. The form is very short, and will take $\approx$ 1 minute to fill out: https://forms.gle/8YnAVCYbb2EV9Cgx7.

   We really appreciate all your feedback and help with making CS161 the best possible learning experience it can be! If you would like to give more general feedback on the class as a whole, please check out the many possible options highlighted in the linked Ed post[4].

   Have you submitted your answers to the google form above?

   [**We are expecting:** *Yes*]

[4] https://edstem.org/us/courses/38246/discussion/2970522