CS 161 **Problem Set 5**
Fall 2025 **Due:** Friday November 14, at 11:59pm on Gradescope.

Style guide and expectations: Please see the top of the "Homework" page on the course 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.

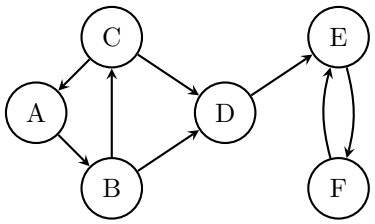
LLM policy: Check out the course webpage for best practices on how to productively use LLMs on homework, if you use them at all.

Exercises

We recommend you do the exercises on your own before collaborating with your group. The point is to check your understanding.

---

1. **(3 pt.)** Consider the following graph G:



   (a) **(1 pt.)** What is the SCC DAG (aka the *SCC meta-graph*) for G?
   [**We are expecting:** *A drawing or extremely clear description of the SCC meta-graph.*]

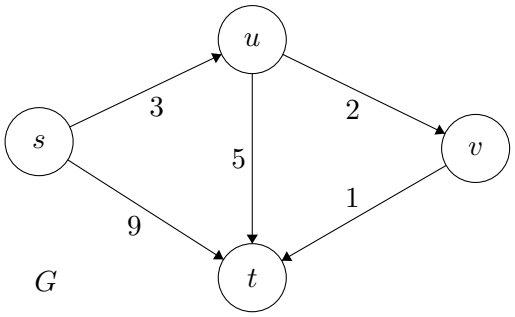
   (b) **(2 pt.)** Recall that Kosaraju's algorithm to identify SCCs does one run of DFS, reverses all the edges, and then does a second run of DFS. If you run Kosaraju's algorithm on G, which SCC in your SCC DAG does the *second* run of DFS start in? (That is, it starts at some vertex v of G. Which meta-vertex of the SCC DAG does v live in?)
   [**We are expecting:** *Clearly identify which SCC the second DFS run must start in. No explanation is required, but it would be good to think about why there is a unique answer to this question, regardless of where you started the first DFS run.*]

2. **(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
    while F isn't empty:
        x = a vertex v in F with minimum d[v]
        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)

    // 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.



continued on next page...

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

- *The table below filled out*
- *The shortest path and its cost that the algorithm returns.*

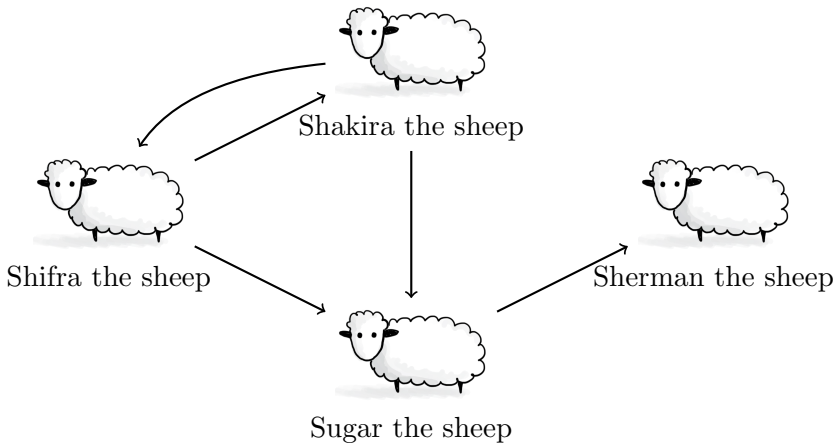
No justification is required.]

	$\mathtt{d}[s]$	$\mathtt{d}[u]$	$\mathtt{d}[v]$	$\mathtt{d}[t]$	$\mathtt{p}[s]$	$\mathtt{p}[u]$	$\mathtt{p}[v]$	$\mathtt{p}[t]$
When entering the first while loop for the first time, the state is:	0	∞	∞	∞	None	None	None	None
Immediately after removing the first element from F, the state is:	0	3	∞	9	None	s	None	s
Immediately after removing the second element from F, the state is:								
Immediately after removing the third element from F, the state is:								
Immediately after removing the fourth element from F, the state is:								

Problems

---

3. **(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 BaahGram (it's a platform for sheep to share photos of scenic pastures, new shearing styles, and so on[1]). 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 this directed, unweighted graph on the 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:*

- *Pseudocode or a very clear English description of your algorithm*

[1] Also my new start-up idea `#bleatthealgorithm`

- *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.)** 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 or 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.]

4. **(5 pt.)** (**Dijkstra with negative edges**) 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 2. 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 (at most 5 vertices)*
- *An explanation of why there is no shortest path from s to t.*

]

(b) **(3 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:**

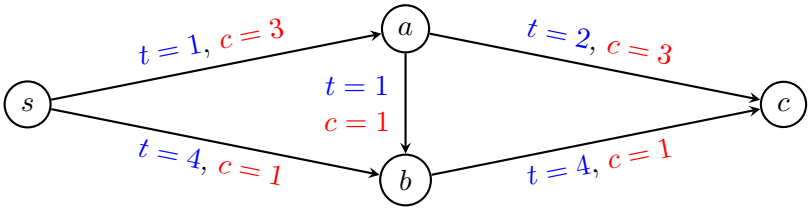
- *A small example (at most 5 vertices)*
- *An explanation of what `Dijkstra_st_path` does on this graph and why it does not return a shortest path.*

]

5. **(10 pt.) [The Shortest Green-Enough Path]** Let $G = (V, E)$ be a directed graph, where each edge (u, v) has *two* weights. It has a weight $t(u, v)$ that represents the time (in minutes) it takes to traverse that edge; and a weight $c(u, v)$ that represents the carbon footprint cost (in kilograms of CO_2) it takes to traverse that edge. Suppose that for all $(u, v) \in E$, $t(u, v)$ and $c(u, v)$ are non-negative integers.

   Your goal is to solve the following version of the single-source shortest-path problem: Given a starting vertex s, and a carbon budget B (a positive integer), find, for all vertices $v \in V$, the shortest path (in terms of time) from s to v *subject to the constraint* that the total cost of the path (in terms of carbon) is *strictly less* than B kg of CO_2.

   For example, let G be the graph below:



   On this graph with a budget of $B = 6$ and starting vertex s, your algorithm should return the following information (in a format of your choosing, it doesn't need to be a table):

Destination	Time (min)
a	1
b	2
c	6

   For example, to get from s to c, we should travel along the path $s \to a \to b \to c$, which takes time 6 minutes and costs 5 kilograms of CO_2. The path $s \to a \to c$ is faster (time only 3 minutes), but the cost is $6 \geq B$ kilograms, which is too large.

   Your algorithm should run in time $O((Bn + Bm)\log(Bn))$, where n is the number of vertices in G and m is the number of edges. You may (and, hint, may want to) use any algorithm we have seen in class as a black box.

   [**Hint:** *Consider modifying $G = (V, E)$ to create a new graph $G' = (V', E')$ with Bn vertices, by replacing each vertex $v \in V$ with B copies of v, $(v, 0), (v, 1), \ldots, (v, B-1) \in V'$. Think of the vertex $(v, j) \in V'$ as "the vertex $v \in V$ in the case that it took j kilograms of CO_2 to get to v from s". Then replace each edge in E with up to B edges in E'... How would you place these edges in a way that is consistent with the hinted interpretation of (v, j) above?*]

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

   - *A clear description of your algorithm;*
   - *An explanation of why it is correct;*
   - *A short justification of the running time.*

   You do not need to give pseudocode, but you may if you think it makes your solution clearer. Your explanation doesn't need to be a formal proof, but it should be clear to the grader.]

6. **(6 pt.) The Shortest Greenest Path (Ethics):** Given your expertise after having completed Problem 5, a ride-sharing company taps you to help reduce its carbon footprint in a particular city. The company represents the city as a weighted graph, where each edge weight represents the expected travel time between intersections. The company also wants to factor in carbon emissions on each route segment, and as in the previous problem they have an estimate of this as well. But when the engineers try to combine the two into a single "cost" to minimize, they struggle to decide how to weigh one against the other. They could do what you did in Problem 5 and impose a total carbon budget B. But is that the "right" thing to do? Is there any "right" way to combine the two costs?

   (a) **(3 pt.)** Using the concept of **incommensurability** from one of the Embedded Ethics lectures, explain why the team has difficulty deciding how to combine travel time and carbon footprint into a single weight.
   [**We are expecting:** *We are expecting: 1–2 sentences explaining why these values are incommensurable.*]

   (b) **(3 pt.)** The Embedded Ethics lecture discusses other real-world examples of algorithms that treat incommensurable values as if they were commensurable, like credit scoring. Give another real-world example where an algorithmic model treats incommensurable values as if they were commensurable. What ethical risks arise when distinct kinds of value are collapsed into one scale?
   [**We are expecting:** *We are expecting: 2–4 sentences describing an example algorithmic model and one risk of value simplification.*]