CS 161 **Problem Set 7**

Fall 2025 **Due:** Friday December 5, 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.

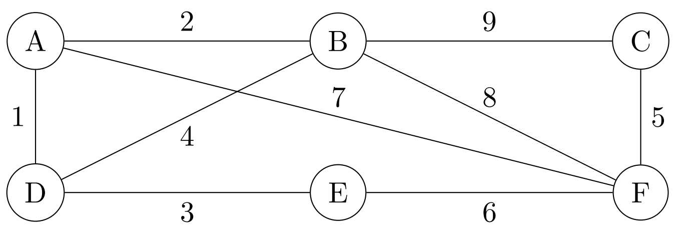
LLM policy: Check out the course webpage for best practices on how to productively use LLMs on homework, if you use them at all.

Exercises

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

---

1. **(2 pt.)** Consider the graph G below.



   (a) **(1 pt.)** In what order does Prim's algorithm add edges to an MST when started from vertex C?

   (b) **(1 pt.)** In what order does Kruskal's algorithm add edges to an MST?

   [**We are expecting:** *For both, just a list of edges. You do not need to draw the MST, and no justification is required.*]

2. **(6 pt.)** In this exercise, we will walk through a greedy algorithm for the following problem. Suppose that a troupe of penguins is hiking along a scenic antarctic hiking trail of length T meters. There are n scenic vistas along the way, located at distances $x_1 < x_2 < \cdots < x_n$ meters. The troupe wants to stop at as many scenic vistas as possible, but wants to space them out so that there are at least k meters between any two stops.

   (a) **(3 pt.)** Design a greedy algorithm that achieves the goals outlined above. Your algorithm should take as input k and the values $x_1, \ldots, x_n$ (in sorted order). It should output a list of stops $i_1 < i_2 < \ldots < i_m$ so that m is as large as possible, subject to the constraint that $x_{i_j} - x_{i_{j-1}} \geq k$ for all $j = 1, \ldots, m$. Your algorithm should take time $O(n)$.

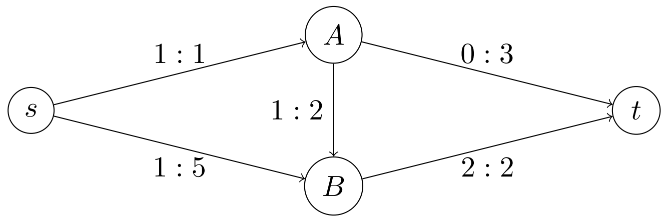
   [**We are expecting:** *A clear explanation of your greedy choices, and a clear English description of your algorithm. You don't need to justify the running time. (You'll prove correctness in part (b)).*]

   (b) **(3 pt.)** In this part, you will prove that your algorithm from part (a) is correct. Following the examples that we saw in class, the structure of the proof is shown below: we have filled in the inductive hypothesis, base case, and conclusion. Fill in the inductive step to complete the proof.

   - **Inductive hypothesis:** After making the t'th greedy choice, there is an optimal way to pick vistas that extends the partial solution the algorithm has constructed so far.
   - **Base case:** Any optimal solution extends the empty solution, so the inductive hypothesis holds for $t = 0$.
   - **Inductive step:** *(you fill in)*
   - **Conclusion:** At the end of the algorithm, the algorithm returns a list S^* of scenic vistas that are all at least k meters apart. By the inductive hypothesis, we know there is an optimal solution extending S^*. Yet, there is no solution extending S^* other than S^* itself, since there are no more vistas that could be added to the end without making two vistas closer than k meters. This implies that S^* is optimal, and hence the algorithm returns an optimal solution.

   [**We are expecting:** *A proof of the inductive step: assuming the inductive hypothesis holds for t − 1, prove that it holds for t.*]

3. **(3 pt.)** *Note: this exercise covers material from Lecture 16, which we will not get to until after the break.* Consider the following graph. The notation $x : y$ means that the edge has flow x and capacity y.



Following the Ford-Fulkerson algorithm, find an augmenting path in this graph, and update the flow accordingly. What is the flow after your update?

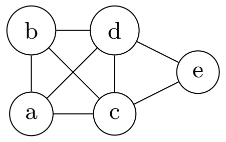
[**We are expecting:** *A description of you augmenting path (of the form* __ $\to$ __ $\to$ __ $\to$ __*), and either a picture or a list of the flows after the update. Also, we are expecting the value of the flow after the update.*]

Problems

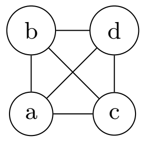
4. **(6 pt.)** [**k-well-connected graphs.**] Let $G = (V, E)$ be an undirected, unweighted graph with n vertices and m edges. For a subset $S \subseteq V$, define the **subgraph induced by** S to be the graph $G' = (S, E')$, where $E' \subseteq E$, and an edge $\{u, v\} \in E$ is included in E' if and only if $u \in S$ and $v \in S$.

   For any $k < n$, say that a graph G is k-well-connected if every vertex has degree at least k.

   For example, in the graph G below, the subgraph G' induced by $S = \{a, b, c, d\}$ is shown on the right. G' is 3-well-connected, since every vertex in G' has degree at least 3. However, G is not 3-well-connected since vertex E has degree 2.



$G = (V, E)$



$G' = (S, E')$, for $S = \{a, b, c, d\}$

   Design a greedy algorithm to find a maximal set $S \subseteq V$ so that the subgraph $G' = (S, E')$ induced by S is k-well-connected. In the example above, if $k = 3$, your algorithm should return $\{a, b, c, d\}$, and if $k = 4$ your algorithm should return the empty set.

   You may assume that your representation of a graph supports the following operations:

   - `degree(v)`: return the degree of a vertex in time $O(1)$
   - `remove(v)`: remove a vertex and all edges connected to that vertex from the graph, in time $O(\text{degree}(v))$.
   - `vertices(G)`: return the list of non-removed vertices in time $O(n)$.

   Your algorithm should run in time $O(n^2)$.

   You do not need to prove that your algorithm works, but you should give an informal (few sentence) justification.

   [**Hint:** *Think about greedily* ***removing*** *vertices.*]

   [**We are expecting:**

   - *Pseudocode* ***AND*** *an English description of what your algorithm is doing.*
   - *An informal justification of the running time.*
   - *An informal justification that the algorithm is correct.*

   ]

5. **(6 pt.) [Badger Badger Badger (part 2)]** The tunnel-digging badger from HW4 is back, along with $n-1$ of their friends! There are n badgers in total. You'd like to get a message to all n of the badgers (perhaps you'd like to tell them your solutions to HW4!). However, each of the badgers pops up out of the ground for only one interval of time. Say that badger i is above ground in the interval $[a_i, b_i]$. Your plan is to shout your message repeatedly, at certain times $t_1, \ldots, t_m$. Any badger who is above ground when you shout your message will hear it, while any badger who is below ground will not. Assume that shouting the message is instantaneous.

   You'd like to ensure that each badger hears your message at least once (it's okay if a badger hears it multiple times), while making m as small as possible (so that you don't have to shout too much).

   Design a greedy algorithm which takes as input the list of intervals $[a_i, b_i]$ each badger is above ground and outputs a list of times $t_1, \ldots, t_m$ so that every badger hears your message at least once and m is as small as possible. Your algorithm should run in time $O(n \log(n))$. If it helps, you may assume that all the a_i, b_j are distinct.

   You do not need to prove that your algorithm is correct. However, we strongly suggest that you at least prove it to yourself—or at least work out the intuition/outline of a proof—to make sure that your algorithm is correct! (Note: The solutions will include a proof so you can check your answer later if you choose to write down a formal proof for more practice).

   [**We are expecting:** *Pseudocode and an English description of the main idea of your algorithm, as well as a short justification of the running time.*]

6. **(5 pt.) [EthiCS: Building the "Best" Emergency Network]** Suppose a region wants to build a network of emergency supply stations for wildfire response. Each town gives a list of nearby towns they would be willing to share supplies with during an emergency. We model this using an undirected weighted graph $G = (V, E)$ where:

   - Each vertex is a town.
   - There is an edge $a, b \in E$ if towns a and b agree to share supplies.
   - The weight on each edge represents the estimated cost of building and maintaining a supply route between them.

   A straightforward way to minimize total cost is to apply a greedy minimum-spanning-tree algorithm (such as Kruskal's algorithm). This algorithm finds a minimum-cost network of supply routes. However, the wildfire response team expresses concerns that the resulting plan is not in the best interests of all towns.

   Using the language developed in our Embedded Ethics lectures (idealization, abstraction, and incommensurability), explain some of the real-world issues with using this greedy algorithm to design an emergency supply network.

   [**We are expecting:** *2-4 sentences identifying real-world issues with this algorithm, explicitly mentioning and applying at least two of idealization, abstraction, incommensurability.*]

7. **(5 pt.) [Uniqueness of MSTs]** Recall the following statement from class:

Lemma 1. *Let G be a connected, weighted, undirected graph with distinct edge weights. Then G has a unique minimum spanning tree.*

We will prove the lemma together.

(a) **(2 pt.)** For sake of contradiction, suppose that T_1 and T_2 are two distinct spanning trees in G, and let $e = \{u, v\}$ be the lowest weight edge that is in exactly one of T_1 and T_2. Without loss of generality, suppose that $e \in T_2$ and $e \notin T_1$.

Now consider adding e to T_1. Prove that this must form a cycle in T_1, and further that this cycle must contain some other edge $f = \{x, y\}$ with weight strictly larger than e.

[**We are expecting:** *A formal proof. Formal proofs can sometimes be short, but they always must fully explain why the statement is true.*]

(b) **(2 pt.)** Let T_1' be the following modification of T_1: We start with T_1, add e, and we remove the edge f from the previous part. Prove that T_1' is a spanning tree.

[**We are expecting:** *A formal proof. Formal proofs can sometimes be short, but they always must fully explain why the statement is true.*]

(c) **(1 pt.)** Derive a contradiction and conclude that T_1 and T_2 couldn't have been distinct MSTs.

[**We are expecting:** *A formal proof. Formal proofs can sometimes be short, but they always must fully explain why the statement is true.*]

8. **(0 pt.)[OPTIONAL: A Band of Busy Beavers]** *Note: This problem relies on Lecture 16, which we will not get to until December 2, the same week HW is due. For that reason, we've made it optional (we realize you probably already have lots of stuff to do in week 10). But we think it's great practice and would encourage you to do it, either during Week 10 or when studying for the exam. This material is fair game for the exam.*

A band of n busy beavers has a n dams to build at various points along the river. They have n bundles of sticks at their disposal to build the dams with. However, there are a number of constraints:

- Each dam requires exactly one beaver and exactly one bundle of sticks to complete.
- Each beaver can only work on one dam (due to time constraints).
- Each bundle of sticks can only be used once.
- Each beaver can only work on particular dams (e.g., the ones near where they happen to be right now). For each beaver i and dam j, let $c_{i,j}$ be 1 if beaver i can work on dam j, and zero otherwise.
- Each bundle of sticks can only be used on particular dams (the ones that happen to be near where the sticks are right now). For bundle k and dam j, let $d_{k,j}$ be 1 if bundle k can be used on dam j, and zero otherwise.

Given these inputs (that is, $c_{i,j}$ for $i = 1, \ldots, n$ and $j = 1, \ldots, n$; and $d_{k,j}$ for $k = 1, \ldots, n$ and $j = 1, \ldots, n$), design an algorithm to assign beavers and stick-bundles to dams in a way that maximizes the number of dams that can be built.

Your algorithm should run in time $O(n^5)$.

[**We are expecting:** *Nothing, this problem is optional. But if we were to ask for something, it would be: A very clear English description of your algorithm, an informal justification of why it is correct, and an informal justification of the running time. Pseudocode is not required, but you may include it if you think it will make your answer clearer. You can (and hint, probably should), use any algorithm we have seen in class as a black box.*]