CSE 421 Winter 2026: Problems

Instructions: Solutions should be legibly handwritten or typeset (ideally in LaTeX). Mathematically rigorous solutions are expected for all problems unless explicitly stated. You are encouraged to collaborate on problems in small teams, but everyone must individually submit solutions. Use of generative AI for assistance on homework needs to adhere to the AI policy found on the course webpage. Additionally, solutions for the problems may be found online or in textbooks—but do not use them.

For grading purposes, on each problem, **list the names of your collaborators**. Additionally, for each problem, if used, **describe how you used generative AI to assist you**. Note that each problem will be assigned an individual Gradescope assignment.

All problems are worth 10 points. Problems numbered < 50 are general problems. Problems numbered ≥ 50 are challenge problems.

If a runtime is not specified, you are expected to come up with the best runtime possible. We will assume graphs have n vertices and m edges by convention.

Due Date	Problems
Fri Jan 16th 6:00 pm	1-3, 51
Fri Jan 23rd 6:00 pm	4-6, 52
Fri Jan 30th 6:00 pm	7-9, 53
Fri Feb 6th 6:00 pm	10, 54
Fri Feb 13th 6:00 pm	11-13, 55
Fri Feb 20th 6:00 pm	14-16, 56
Fri Feb 27th 6:00 pm	17-19, 57
Fri Mar 6th 6:00 pm	20-22, 58
Fri Mar 13th 6:00 pm	23-25, 59

Grading Your course grade will be calculated using the following formula:

$$\begin{aligned} \text{Grade} &= 0.56 \cdot g + 0.04 \cdot c + 0.15 \cdot m + 0.25 \cdot f \\ \text{where } g &= \frac{1}{220} \cdot [\text{sum of best 22 general problems}], \\ c &= \frac{1}{70} \cdot [\text{sum of best 7 challenge problems}], \\ m &= [\text{midterm score}], \text{ and } f\,[\text{final score}]. \end{aligned}$$

We may apply a curve of the form $x \mapsto \frac{\log(1+\kappa x)}{\log(1+\kappa)}$ to exam scores with $\kappa \approx 2$. A grade of ≥ 0.92 guarantees a 4.0. A grade of ≥ 0.75 guarantees at least a 3.0.

Contents

Part I

General problems

1 Big O notation

1. **[5 points]** Prove or disprove the following statement: There exist *monotonically increasing* functions $f, g : \mathbb{N} \to \mathbb{N}$ such that f is not $O(g)$ *and* g is not $O(f)$. If it is true, give an example. If it is false, prove that no such functions exist. For partial credit, you can find functions f and g that are not monotonically increasing.

2. **[5 points]** Let $f_1, f_2, \ldots, f_k : \mathbb{N} \to \mathbb{N}$ be a set of functions. Show there exists a function $g : \mathbb{N} \to \mathbb{N}$ with the following properties:

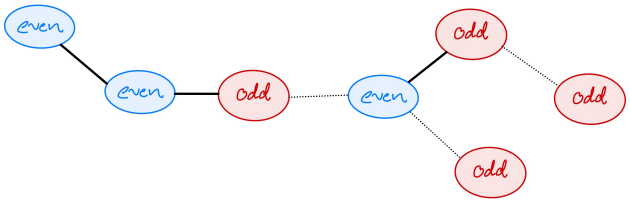
   (a) f_i is $O(g)$ for each $i = 1, \ldots, k$.

   (b) For any function $g' : \mathbb{N} \to \mathbb{N}$ such that f_i is $O(g')$ for each $i = 1, \ldots, k$, it follows that g is $O(g')$.

   In other words, there exists an asymptotic least upper bound for any finite collection of functions.

2 Tree induction

Given a connected tree T with $n \geq 2$ vertices, suppose each vertex of the tree is marked with either "odd" or "even" with an even number of vertices marked as "odd". Prove there exists a subset F of the edges of T such that each "odd" marked vertex is adjacent to an odd number of edges in F and every "even" marked vertex is adjacent to an even number of edges in F. In the example below, the dashed edges would form a valid F.



Hint: For a proof by induction, you can first prove the statement for all graphs of $n = 2$ vertices. Then, for the inductive step, when trying to prove the statement for a graph of n vertices, you can assume the statement is true for *all* graphs of $< n$ vertices.

3 Back edges

A DFS tree is the tree constructed by running the DFS algorithm and recording which edges are used in the search. *We define, for this problem*[1]*, a back edge is any edge of the original graph from a vertex to its ancestor in the tree (including itself and its parent).* Prove that if a directed connected graph has a back edge with respect to any DFS tree, then it has a directed cycle. Second, prove that every directed cycle in a directed graph must include at least one back edge with respect to any DFS tree.

4 Bug classification

An entomologist has discovered a new collection of bugs. However, she notices that if two bugs are placed in a terrarium together, they either are hostile to each other or not. She hypothesises that she has actually discovered two species that are visually indistinguishable but are docile with other members of the same species and hostile to members of the other species.

Without loss of generality, they have decided that Bug 1 is of Species A. Her graduate students have placed various pairs of bugs (x_i, y_i) in the terrarium and recorded their interaction as either hostile or docile.

Given a list of m data points $(x_i, y_i, \text{hostile/docile})$ consisting of n bugs, **construct** an algorithm which outputs either:

1. An assignment of each bug $1 \ldots n$ as either species A, species B, or "cannot determine". The "cannot determine" label should be given if we cannot definitively mark the Bug as species A or B.
2. An output of "hypothesis false" if the collected data is inconsistent with the hypothesis.

Additionally, **prove** the runtime and correctness of your algorithm.

Note: The hypothesis could be proven false, even if some bug's species cannot be concretely determined.

(Optional): Let's say we decided to avoid the "cannot determine" label altogether, and instead consider labels that assign only "species A" or "species B" to each bug. Now, assuming the hypothesis is true, we are interested in labelings that are not demonstrably incorrect (i.e., labelings that do not directly contradict the data). Let's refer to these as consistent labelings. Modify your algorithm so that, when it doesn't output "hypothesis false", it outputs the number of consistent labelings.

5 Graph disconnector

Suppose an undirected graph $G = (V, E)$ with $n = \|V\|, m = \|E\|$ has two vertices s and t that are more than $> n/2$ apart (i.e., the shortest distance from s to t is strictly more than half the number of vertices

[1]The literature, including the textbooks for this course, tends to vary in its definition of a back edge. Some literature will call a self-loop from $v \to v$ a back edge. Some will not. Some literature will distinguish this case from other ancestors by referring to any edge to a parent or further back as a proper ancestor. I, too, often confuse the definitions. We will attempt to make it clear in our problem statements, like so.

in the graph). Prove that there is some third vertex v so that if you remove v, the vertices s and t are now disconnected.

Using your proof as inspiration, modify an algorithm you have seen from class to construct an algorithm for finding such a vertex v. State the runtime of your algorithm and use your proof to argue correctness.

Hint: Let ℓ be the distance between s and t. First, argue that any path from s to t contains at least one vertex at each of the distances $0, 1, 2, \ldots, \ell$ from s.

6 Shortest paths with multiple goals

A bakery needs to procure k distinct ingredients for the day's breadmaking. The ingredients are available at various stores in town, with each store pricing the items differently. Due to refrigeration constraints, the bakery needs to procure the items in order: first item 1, then item 2, ..., then item k.

The bakery models the problem as a *directed* graph $G = (V, E)$ with non-negative weights $w : E \to \mathbb{R}_{\geq 0}$ and a matrix $f : V \times [k] \to \mathbb{R}$ with $f(v, i)$ describing the cost of buying item i at vertex v. The weight of each edge $u \to v$ describes the cost of driving from u to v. Additionally, if the item i is unavailable at vertex v, then $f(v, i) = \infty$.

The bakery starts their procurement truck at vertex s and their procurement truck should return to s after retrieving all k items.

Give an algorithm along with its proof of runtime and correctness that finds the *minimal* cost route that retrieves all k items. Your algorithm should output a sequence of edges traversed and items bought. For full credit on this problem, you should produce an algorithm with an asymptotic runtime[2] of $km \log(kn)$ or better! Assume the input is the graph G expressed as an adjacency list.

Hint: How can we express the "state" of the truck at any given time? In most graph problems, the state of the truck would be a vertex $v \in V$. What is it for this problem?

(Optional) What if there were no refrigeration constraints and the items could be bought in any order? What would be the runtime of a similar algorithm in this scenario?

7 Scheduling with penalties

You are given n tasks. Task i requires $\tau_i \in \mathbb{N}$ units of processing time and incurs a penalty $p_i > 0$ for every unit of time it waits before being completed. Tasks are processed on a single machine that can process at most one task at a time.

[2]The road network dataset for the USA has approximately $n = 2 \cdot 10^7$ vertices and $m = 3 \cdot 10^7$ edges.

Let C_i denote the completion time of task i in the schedule. The total penalty of a schedule is $\sum_{i=1}^{n} p_i \cdot C_i$.

1. Show that sorting the tasks in increasing order of processing time τ_i does not always produce a penalty minimizing schedule.
2. Find a different objective function for a greedy algorithm for penalty minimization. Prove correctness and give runtime.

Hint: For part (2), consider two tasks scheduled consecutively in the "wrong" order and show that swapping them strictly improves the objective value.

8 Assignment with thresholds

Assume there are n items with values $v_1, \ldots, v_n \in \mathbb{Z}_{\geq 0}$. Assume there are n customers with requirement thresholds $r_1, \ldots, r_n \in \mathbb{Z}_{\geq 0}$. Say an assignment of items to customers is *valid* if

1. The assignment is a matching: Each item is assigned to at most 1 customer, and each customer is assigned at most 1 item.
2. If item i is assigned to customer j then $v_i \geq r_j$.

Find the *valid* assignment of items to customers that maximizes the number of items that are assigned. Prove your algorithm is correct and analyze its runtime.

Hint: For a helpful starting point, cook up some small $n \approx 5$ sized examples and draw the optimal assignment/matching. What do you notice about it?

9 Adjusting MSTs

Consider a weighted undirected connected graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$. Let T be *a* minimum spanning tree for G.

Design a $O(m)$ runtime algorithm for *deciding* if the minimum spanning tree needs updating when an edge $e \in E$ has its weight adjusted from $w(e)$ to w'. You can assume the edge is between distinct vertices (u, v). Prove runtime and correctness. The algorithm does not need to calculate the new MST. You only need to *decide* if the MST needs updating.

Hint: Is there a theorem from the lectures that could help identify if the MST needs updating?

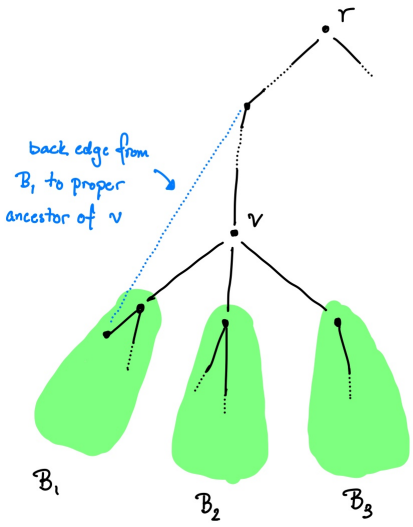
Part II
Challenge problems

51 Articulation points

For an undirected connected graph $G = (V, E)$, a vertex v is defined as an **articulation point** if the removal of v and its adjacent edges *disconnects* the graph into two or more connected components. We will construct a $O(n + m)$-time algorithm for finding *all* the articulation points of a connected graph.

1. Let G be a connected graph with at least two vertices. Let T be a DFS tree of G rooted at r. Prove that r is an articulation point iff r has at least two children in T.

2. Let v be a vertex in the graph and let $B_1, \ldots, B_k$ be the subtrees of the DFS tree T descending from v. Prove that v is **not** an articulation point iff for each subtree B_i, there exists a back edge from B_i to a proper ancestor[3] of v in T.



3. Enumerate the vertices, $\#(v)$, in the order they are visited by the DFS algorithm generating T from root r. For any vertex v, let B_v be the subtree of T rooted at v. Define $\text{low}_T(v)$ as the minimum of $\#(v)$ and $\#(u)$ for any proper ancestor u of v connected to B_v by a single back edge.

   Construct an algorithm[4] for computing $\text{low}_T(v)$ for the DFS tree T rooted at r that runs in time $O(n + m)$. Prove runtime and correctness.

[3] A proper ancestor of v is a vertex along the path between v and r in T not including v.

[4] You do not need to reinvent the wheel here. If your algorithm is a minor modification of a known algorithm such as DFS, you can simply write, "We modify DFS to calculate [blank] after we have traversed its children."

4. Having computed $\text{low}_T(v)$ for every vertex for a DFS tree rooted at T, use the property proven in Part 2 to compute all the articulation points of the graph in time $O(n + m)$. Prove runtime and correctness.

52 Independent set

For an undirected graph $G = (V, E)$, an independent set $V' \subset V$ is a set of vertices such that there are no edges between elements of V'. A computational task of interest is finding the largest possible independent set within a graph G. Today, we construct a greedy algorithm for finding an independent set of size $\geq n/(d+1)$ where $d = 2|E|/|V|$ is the average degree.

The high-level idea of the algorithm is fairly simple and should remind you of examples we have observed in class: Given the input undirected graph, pick the vertex of smallest degree to add to the independent set, prune[5] it and its neighbors from the graph, and repeat.

1. Give an example of a graph where the greedy algorithm fails to find the optimal independent set.
2. Spell out the algorithm in more detail using better data structures and prove its runtime.
3. Analyzing a greedy approximation algorithm is something you have only seen a few times. We'll walk you through another analysis here. Let's observe that the algorithm terminates once all vertices are pruned. Let $I \subseteq V$ be the independent set produced by this greedy algorithm. For $v \in I$, define $f(v)$ to be the number of vertices (including v) that are pruned when v is added to I.
   (a) What is an equation comparing n and $\{f(v)\}_{v \in I}$?
   (b) In terms of $f(v)$, give a *lower bound* for the sum of the degrees of all the vertices pruned when v is added to I.
   (c) Express this as an inequality relating d, the average degree, and the set of values $\{f(v)\}_{v \in I}$.
   (d) Apply the following property:

$$\left(\sum_v f(v) \right)^2 \leq |I| \sum_v f(v)^2.$$

   This is the Cauchy-Schwarz inequality and can be proven by expanding the statement

$$\sum_{v,w \in I} (f(v) - f(w))^2 \geq 0.$$

   (In your solution, you can assume the Cauchy-Schwarz inequality for free).
   (e) Now conclude that $|I| \geq n/(d+1)$.

[5] Pruning a vertex from a graph means removing it and all its associated edges.

53 Profit-based job scheduling

In this variant of the job scheduling problem, there are n jobs each with an integer deadline $d_i \in \{1, \ldots, n\}$ and a profit $p_i > \mathbb{R}_{>0}$. Each job takes unit time to complete, and the profit earned per job is p_i if the job's termination time t_i is $\leq d_i$ and the profit earned is 0 if the job's termination time t_i is $> d_i$. The earliest start time for any job is 0.

Construct an algorithm that outputs the termination times for each of the jobs in a profit-maximizing schedule. It is okay to answer ∞ for the termination time for a job, indicating that you do not intend to undertake said job. Assume that the arithmetic of deadlines and profits has $O(1)$ time and space complexity. Provide proofs of runtime and correctness.

For 8 points, your algorithm should run in time $O(n^2)$. For full credit, your algorithm should run in time $O(n \log n)$.

Hints and suggestions:

1. One step in your argument will be showing that any optimal solution assigns only integer termination times. Convince yourself this is true. To make your life and our grading easier, you don't need to give a proof of this statement in your answer. You may assume it for free.
2. Like many examples in class, a good place to start a proof of correctness is to consider the **first** time your algorithm and a strategy deviate.
3. In our solution, there was some casework required to prove our algorithm's correctness.