CS/ECE 374 A ✧ Fall 2025
Final Exam Problem 1 Solution

For each statement below, there are two boxes in the answer booklet labeled "Yes" and "No". Check "Yes" if the statement is ***always*** true and "No" otherwise, and give a ***brief*** (at most one short sentence) explanation of your answer. **Assume $P \neq NP$.** If there is any other ambiguity or uncertainty about an answer, check "No".

Read each statement *very* carefully; some of these are deliberately subtle!

(a) Which of the following statements are true for ***every*** language $L \subseteq \{0, 1\}^*$ and every ***regular*** language $R \subseteq \{0, 1\}^*$?

- If $L \cup R$ is regular, then L is regular.

  Yes | ~~No~~ X — Consider $L = \text{HALT}$ and $R = \Sigma^*$

- If $L \cup R$ is not regular, then L is not regular.

  ~~Yes~~ X | No — The union of any two regular languages is regular.

- If $L \cap R$ is regular, then L is regular.

  Yes | ~~No~~ X — Consider $L = \text{HALT}$ and $R = \varnothing$

- If there is a reduction from L to R, then $L \in P$.

  Yes | ~~No~~ X — You didn't say "polynomial-time"!

- If $L \cap R$ is not decidable, then L is not decidable.

  ~~Yes~~ X | No — The intersection of any two decidable languages is decidable.

Problem 1 continues onto the next page.

(b) Which of the following statements are true for ***every*** language $L \subseteq \{0, 1\}^*$?

- L^* is regular.

  Yes ~~No~~ Consider $L = \left\{0^{2^n}1 \mid n \geq 0\right\}$.

- L^* contains the empty string ε.

  ~~Yes~~ No By definition of L^*.

- If L is NP-hard, then L^* is NP-hard.

  Yes ~~No~~ Consider $0 + 1 + 3\textsc{Color}$.

- If L is a fooling set for an NP-hard language, then L is infinite.

  Yes ~~No~~ $\varnothing$ is a fooling set for *every* language.

- If L is the intersection of two NP-hard languages, then L is NP-hard.

  Yes ~~No~~ If A is NP-hard, then $\Sigma^* \setminus A$ is also NP-hard, but $A \cap (\Sigma^* \setminus A) = \varnothing$ is not.

CS/ECE 374 A ✧ Fall 2025

Final Exam Problem 2 Solution

For each statement below, there are two boxes in the answer booklet labeled "Yes" and "No". Check "Yes" if the statement is ***always*** true and "No" otherwise, and give a ***brief*** (at most one short sentence) explanation of your answer. **Assume *P* ≠ *NP*.** If there is any other ambiguity or uncertainty about an answer, check "No".

(a) Which of the following languages can be proved undecidable ***using Rice's Theorem***?

- $\{\langle M\rangle \mid M \text{ has at most 374 states}\}$

  Yes | ~~No~~ (checked) — This language is decidable!

- $\{\langle M\rangle \mid M \text{ accepts a finite number of strings}\}$

  ~~Yes~~ (checked) | No — Only about the accepting language of M.

- $\{\langle M\rangle \mid M \text{ accepts } \texttt{KALM}\text{, rejects } \texttt{PANIK}\text{, and hangs on } \texttt{STONKS}\}$

  Yes | ~~No~~ (checked) — Not just about the accepting language of M. `d i s p l e a s e m e n t`

- $\{\langle M\rangle \mid \text{There are exactly 374 palindromes that } M \text{ does } \textit{not} \text{ accept}\}$

  ~~Yes~~ (checked) | No — Only about the accepting language of M.

- $\{\langle M\rangle \mid \langle M\rangle \text{ is a palindrome}\}$

  Yes | ~~No~~ (checked) — This language is decidable!

Problem 2 continues onto the next page.

(b) Suppose we want to prove that the following language is undecidable.

$$\textsc{Hunter} := \left\{\langle M\rangle \;\middle|\; M \text{ accepts } \texttt{RAMEN} \text{ and } \texttt{FANS} \text{ but does not accept } \texttt{DEMONS}\right\}$$

Your mentor Celine suggests a reduction from the standard halting language

$$\textsc{Halt} := \left\{(\langle M\rangle, w) \;\middle|\; M \text{ halts on input } w\right\}.$$

Specifically, suppose there is a machine GOLDEN that decides HUNTER. Celine claims that the following algorithm decides HALT.

DECIDEHALT($\langle M\rangle, w$):
  Write code for the following algorithm:
    COUCH(x):
      if x = DEMONS
        return FALSE
      else
        ⟨⟨*run M on w and return result*⟩⟩
        return $M(w)$
  return GOLDEN($\langle$COUCH$\rangle$)

Which of the following statements must be true for ***all*** inputs $(\langle M\rangle, w)$?

- If M accepts w, then COUCH rejects DEMONS.

  [X] Yes [ ] No — COUCH *always* rejects DEMONS.

- If M accepts w, then GOLDEN accepts $\langle$COUCH$\rangle$.

  [X] Yes [ ] No — COUCH accepts every string but DEMONS, so $\langle$COUCH$\rangle \in$ HUNTER.

- If M diverges on w, then GOLDEN rejects $\langle$COUCH$\rangle$.

  [X] Yes [ ] No — COUCH diverges on every string but DEMONS, so $\langle$COUCH$\rangle \notin$ HUNTER.

- DECIDEHALT decides the language HALT. (That is, Celine's reduction is correct.)

  [ ] Yes [X] No — If M rejects w, then M halts on w, but COUCH rejects every string, so $\langle$COUCH$\rangle \notin$ HUNTER, so DECIDEHALT rejects $(\langle M\rangle, w)$.

- We could instead prove HUNTER is undecidable using Rice's theorem.

  [X] Yes [ ] No — Only about the accepting language of M

CS/ECE 374 A ✧ Fall 2025

Final Exam Problem 3 Solution

Exactly one of the following languages is regular. Which one?

(a) $\{0^a 1^b 0^c \mid a + b = c\}$

(b) $\{0^a 1^b 0^c \mid a + b \equiv c \pmod 2\}$

Indicate which one of these two languages is regular. Describe a DFA or NFA that accepts the regular language, and ***prove*** that the other language is not regular. (You do not need to prove that your DFA or NFA is correct.)

Solution: ***L_b is regular.***

(a) Let $F = 1^* = \{1^n \mid n \geq 0\}$.

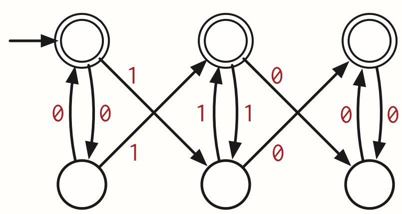
Fix any two strings $x, y \in F$. Then $x = 1^i$ and $y = 1^j$ for some $i \neq j$.

Let $z = 0^i$.

- $xz = 0^0 1^i 0^i \in L_a$ because $0 + i = i$
- $yz = 0^0 1^j 0^i \notin L_a$ because $0 + j = j \neq i$

We conclude that F is a fooling set for L_a. Because F is infinite, L_a cannot be regular.

(b) $L_b = 0^* 1^* 0^* \cap ((0 + 1)(0 + 1))^*$



(All missing transitions go to a hidden dump state.)

■

CS/ECE 374 A ✧ Fall 2025
Final Exam Problem 4 Solution

Suppose we are given two unsorted arrays $A[1..n]$ and $B[1..n]$. Elements of A and B come from some fixed totally ordered set, like integers or letters of the alphabet, that can be compared in $O(1)$ time. An ***upside-down pair*** for A and B is an ordered pair (i, j) of indices such that $i < j$ and $A[i] > B[j]$.

Describe and analyze an efficient algorithm that either finds one upside-down pair for two given arrays A and B, or correctly reports that no such pair exists.

Solution: Following the incredibly subtle hint, we use a divide-and-conquer algorithm.

- If $n = 1$, immediately return FAIL.
- Compute $A[i] = \max A[1..n/2]$ and $B[j] = \min B[n/2+1..n]$ in $O(n)$ time. If $A[i] = B[j]$, return (i, j).
- Recursively look for an upside-down pair for $A[1..n/2]$ and $B[1..n/2]$.
- Recursively look for an upside-down pair for $A[n/2+1..n]$ and $B[n/2+1..n]$.
- If either recursive search finds an upside-down pair, return that pair; otherwise, return FAIL.

The running time satisfies the standard mergesort recurrence $T(n) = O(n) + 2T(n/2)$, so the algorithm runs in $\boldsymbol{O(n \log n)}$ ***time***. ■

Rubric: 10 points; standard divide and conquer rubric. This is not the only $O(n \log n)$-time solution. A brute-force $O(n^2)$-time algorithm is (tentatively) worth 4/10.

Solution (+2 extra credit): The following algorithm runs in $\boldsymbol{O(n)}$ ***time***:

```
HELLFIRECLUB(A[1..n], B[1..n])
  maxAval ← ∞
  maxAind ← −1
  for i ← 1 to n
      if B[i] < maxAval
          return (maxAind, i)
      if A[i] > maxAval
          maxAval ← A[i]
          maxAind ← i
  return FAIL
```

■

Rubric: max 12 points. This is not the only $O(n)$-time solution.

CS/ECE 374 A ✧ Fall 2025
Final Exam Problem 5 Solution

Submit a solution to ***exactly one*** of the following problems (either (a) or (b)).

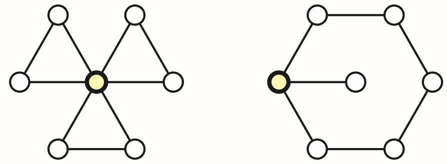
(a) An *ultra-Hamiltonian tour* in a graph G is a closed walk W that visits every vertex in G exactly once, except for *at most* one vertex that W visits more than once.

  i. Give an example of a graph that contains a ultra-Hamiltonian tour, but does not contain a Hamiltonian cycle (which visits every vertex exactly once).

  ii. ***Prove*** that it is NP-hard to determine whether a given graph contains a ultra-Hamiltonian tour.

Solution:

i. Here are two examples. In each of these graphs, every ultra-Hamiltonian tour visits the bold vertex more than once.



ii. We reduce from the standard Hamiltonian cycle problem. Assume G has at least three vertices; otherwise G cannot have a Hamiltonian cycle. Given a graph $G = (V, E)$, we construct a new graph G' as follows.

- Choose an arbitrary vertex $v \in V$.
- Replace v with three vertices $v^\flat$, $v^\natural$, $v^\sharp$.
- Replace each edge uv with edges $uv^\flat$ and $uv^\sharp$.
- Add one more vertex z and three edges $v^\flat v^\natural$ and $v^\natural z$ and $v^\natural v^\sharp$.

$\Longrightarrow$ Suppose G has a Hamiltonian cycle C. This cycle must contain two edges $u{\to}v{\to}w$ through our chosen vertex v. Replacing those two edges with $u{\to}v^\flat{\to}v^\natural{\to}z{\to}v^\natural{\to}v^\sharp{\to}w$ gives us a closed walk in G' that visits $v^\natural$ twice and every other vertex once. So G' has an ultra-Hamiltonian your.

$\Longleftarrow$ Suppose G' has an ultra-Hamiltonian tour W. Then W must visit z, and therefore must contain the subwalk $v^\natural{\to}z{\to}v^\natural$. The only other two neighbors of $v^\natural$ are $v^\flat$ and $v^\sharp$, so W contains the subwalk $u{\to}v^\flat{\to}v^\natural{\to}z{\to}v^\natural{\to}v^\sharp{\to}w$ (or its reversal) for some vertices u and w of G. Vertices u and w must exist and must be distinct, since otherwise G would have fewer than 3 vertices. Replacing that subwalk of W with $u{\to}v{\to}w$ yields a closed walk in G that visits every vertex exactly once. So G has a Hamiltonian cycle.

■

Rubric: 10 points: standard NP-hardness rubric. These are not the only correct solutions to either subpart. No penalty for implicitly assuming $V \geq 100$.

We will only grade one of 5(a) and 5(b). If you submit solutions for both, we will flip a coin to decide which one to grade.

(b) ***Prove*** that the following problem is NP-hard: Given an undirected graph $G = (V, E)$ and a subset of vertices $L \subset V$, does G have a spanning tree T such that every leaf of T is in L?

Solution: We reduce from the Hamiltonian path problem.

Given a graph $G = (V, E)$, we construct a new graph G' by adding two vertices s and t and edges sv and tv for all $v \in V$, and we define $L = \{s, t\}$. Building G' and L from G in polynomial time is straightforward.

$\Longrightarrow$ If G contains a Hamiltonian path P from u to v, then $P + us + vt$ is a spanning tree of G' whose only leaves are s and t.

$\Longleftarrow$ Suppose G' contains a spanning tree T' whose leaves are in L. Because G' has more than one vertex, T' has more than one vertex and thus at least two leaves. Thus, s and t are the leaves of T', which means T' is actually a (Hamiltonian) *path*. It follows that $T' - s - t$ is a Hamiltonian path in G.

■

Rubric: 10 points: standard NP-hardness rubric. This is not the only correct solution.

We will only grade one of 5(a) and 5(b). If you submit solutions for both, we will flip a coin to decide which one to grade.

CS/ECE 374 A ✧ Fall 2025
Final Exam Problem 6 Solution

Describe and analyze an algorithm to compute the maximum total profit you can earn by buying selling, and holding stock for n days, given an array $Price[1..n]$ of stock prices as input. *(See the question handout for a detailed description of this problem.)*

Solution (dynamic programming): For any integers i and k, let $MaxProfit(i,k)$ denote the maximum profit we can earn on days i through n, if we start day i owning k shares of stock. We need to compute $MaxProfit(1,0)$. This function obeys the following recurrence:

$$MaxProfit(i,k) = \begin{cases} 0 & \text{if } i > n \\ \max\begin{Bmatrix} MaxProfit(i+1,k) \\ MaxProfit(i+1,k+1) - Price[i] \end{Bmatrix} & \text{if } i \le n \text{ and } k = 0 \\ \max\begin{Bmatrix} MaxProfit(i+1,k-1) + Price[i] \\ MaxProfit(i+1,k) \\ MaxProfit(i+1,k+1) - Price[i] \end{Bmatrix} & \text{otherwise} \end{cases}$$

(The three options in the last case correspond to selling, holding, and buying on day i.)

We can never own more than n shares of stock, so we can memoize this function into a two-dimensional array $MaxProfit[1..n, 1..n]$. In fact, the optimal strategy always leaves us with zero shares at the end. implies that we never own more than $n/2$ shares. (This requires a proof!) So we only need $n/2$ columns instead of n. (But whatever; that's just a constant factor.) We can fill this array with two nested loops, decreasing i in the outer loop and considering k in any order in the inner loop, in $\boldsymbol{O(n^2)}$ ***time***. ■

Rubric: 10 points: standard dynamic programming rubric. This is not the only correct dynamic-programming solution.

There is an alternative solution are on the next page.

Solution (graph reduction): We construct a directed graph $G = (V, E)$ with weighted edges as follows:

- $V = \{0, 1, 2, \ldots, n\} \times \{0, 1, 2, \ldots, n\}$. Each vertex (i, j) denotes ending on day i owning j shares of stock.
- E is the union of three subsets $E_{\text{buy}} \cup E_{\text{hold}} \cup E_{\text{sell}}$ defined as follows:
  - $E_{\text{buy}} = \big\{(i,j)\to(i+1, j+1) \bigm| 0 \le i \le n-1 \text{ and } 0 \le j \le n-1\big\}$.
    Each edge $(i,j)\to(i+1, j+1)$ has weight $-\mathit{Price}[i]$.
  - $E_{\text{hold}} = \big\{(i,j)\to(i+1, j) \bigm| 0 \le i \le n-1 \text{ and } 0 \le j \le n\big\}$.
    Each of these edges has weight 0.
  - $E_{\text{sell}} = \big\{(i,j)\to(i+1, j-1) \bigm| 0 \le i \le n-1 \text{ and } 1 \le j \le n\big\}$.
    Each edge $(i,j)\to(i+1, j-1)$ has weight $+\mathit{Price}[i]$.

G' is a dag, because each edge $(i,j)\to(i+1,j')$ goes forward in time. We need to compute the longest path in G' from $(0,0)$ to any node (n, j). We can compute all longest paths from $(0,0)$ using a single call to LongestPath in $O(V+E) = \boldsymbol{O(n^2)}$ ***time***.

In fact, the optimal strategy always leaves us with zero shares after day n. (This requires a proof!) So we only need the longest path from $(0,0)$ to $(n,0)$. But finding one longest path is no faster than finding all longest paths from a single source, so whatever. ■

Rubric: 10 points: standard graph reduction rubric. This is not the only correct graph reduction solution.

CS/ECE 374 A ✧ Fall 2025
Final Exam Problem 7 Solution

Suppose you are given a directed graph $G = (V, E)$, where each edge has a positive weight, two vertices s and t, and an integer k. Describe an algorithm that computes the length of the shortest walk in G from s to t that traverses ***at least*** k edges. Analyze your algorithm in terms of V, E, and k.

Solution: Given the input graph $G = (V, E)$, we build a new layered graph $G' = (V', E')$ as follow:

- $V' = V \times \{0, 1, 2, \ldots, k\}$. Each vertex (v, i) means we are at vertex v after traversing exactly i edges if $i < k$, and at least k edges if $i = k$.
- E' is the union of two sets of edges, one connecting vertices in adjacent layers, and one connecting vertices at level k.

$$\begin{aligned} E' = &\Big\{(u,i)\to(v,i+1) \,\Big|\, u\to v \in E \text{ and } 0 \le i \le k-1\Big\} \\ &\cup \Big\{(u,k)\to(v,k) \,\Big|\, u\to v \in E\Big\} \end{aligned}$$

Each edge $(u, i)\to(v, j) \in E'$ has weight $w(u\to v)$.

We need to compute the shortest path in G' from $(s, 0)$ to (t, k). We can do this using Dijsktra's algorithm in $O(E' \log V') = \boldsymbol{O(kE \log kV)}$ ***time***. ■

Rubric: 10 points: standard graph reduction rubric. This is not the only correct solution.

There is an alternative solution are on the next page.

Solution (+2 extra credit): We solve the problem in two phases. To simplify analysis, I will assume that s can reach every vertex in G; otherwise, we can identify all reachable vertices in $O(E)$ time using whatever-first search. This assumption implies $E \geq V - 1$ and therefore $V = O(E)$.

First, given the input graph $G = (V, E)$, we build a new layered graph $G_1 = (V_1, E_1)$ as follow:

- $V_1 = V \times \{0, 1, 2, \ldots, k\}$. Each vertex (v, i) means we are at vertex v after traversing exactly i edges if $i < k$, and at least k edges if $i = k$.
- $E_1 = \left\{(u, i){\rightarrow}(v, i+1) \;\middle|\; u{\rightarrow}v \in E \text{ and } 0 \leq i \leq k-1\right\}$. Each edge $(u, i){\rightarrow}(v, i+1)$ has weight $w(u{\rightarrow}v)$.

G_1 is a dag, because each edge $(u, i){\rightarrow}(v, i+1)$ goes from an earlier layer to a later layer.

For each vertex $v \in V$, let $\mathit{distk}(v)$ denote the length of the shortest path in G_1 from $(s, 0)$ to (v, k); this is also the length of the shortest walk in G from s to v that traverses *exactly* k edges. We can compute $\mathit{distk}(v)$ for every vertex v, using a single call to DagSSSP, in $O(V_1 + E_1) = O(kV + kE) = O(kE)$ time.

Now we construct a second graph $G_2 = (V_2, E_2)$ as follows:

- $V_2 = V \cup \{\mathit{skip}\}$, where *skip* is a new source vertex.
- $E_2 = E \cup \{\mathit{skip}{\rightarrow}v \mid v \in V\}$. Each edge $\mathit{skip}{\rightarrow}v$ has weight $\mathit{distk}(v)$; every other edge $u{\rightarrow}v \in E$ has the same weight in G_2 as it does in G.

Now we need to compute the shortest path in G_2 from *skip* to t. We can compute this shortest path using a single call to Dijsktra's algorithm, in $O(E_2 \log V_2) = O(E \log V)$ time.

The overall algorithm runs in $\boldsymbol{O(kE + E \log V)}$ time. ■

Rubric: 12 points: standard graph reduction rubric; scale partial credit. This is not the only correct algorithm with this running time.