UNIQNAME (print): _____

EECS 376 Final Exam, Winter 2024

Instructions:

This exam is closed book, closed notebook. No electronic devices are allowed. You may use two 8.5×11-inch study sheets (both sides) that you prepared yourself. The last few pages of the exam are scratch paper; you may not use your own. Make sure you are taking the exam at the time slot and location you were assigned by the staff. ***Please print your UNIQNAME at the top of each page.***

Any deviation from these rules may constitute an honor code violation. In addition, the staff reserves the right **not** to grade an exam taken in violation of this policy.

The exam consists of **12 multiple-choice** questions, **3 short-answer** questions, and **2 longer-answer** questions. For the short- and longer-answer sections, please write your answers clearly in the spaces provided. If you run out of room or need to start over, you may use the blank pages at the end, but you **MUST** make that clear in the space provided for the answer. The exam has 17 pages printed on both sides, including this page and the blank pages at the end.

Leave all pages stapled together in their original order.

Sign the honor pledge below.

Pledge:

I have neither given nor received aid on this exam, nor have I concealed any violations of the Honor Code.

I will not discuss the exam with anyone until noon on Thursday May 2nd (once every student in the class has taken the exam, including alternate times).

I attest that I am taking the exam at the time slot and the location I was assigned by the staff.

Signature: _____

Print clearly:

Full Name: _____

Uniqname: _____

This page is intentionally left blank.

UNIQNAME (print): _____

Multiple Choice: Select the one correct option

For each of the problems in this section, select **the one** correct option. Each one is worth 3 points; no partial credit is given.

1. Select the **correct description** for the following statement: assuming that $\mathsf{P} \neq \mathsf{NP}$, the language

   $$\textsc{VertexCoverBig} = \{G = (V, E) : G \text{ is an undirected graph with a vertex cover of size } |V| - 1\}$$

   is NP-complete.

   ◯ The statement is true.

   ● **The statement is false.**

   ◯ It is unknown whether the statement is true or false.

   Solution: This language is in P because a graph has a vertex cover of size $|V| - 1$ if and only if it is *not* a complete graph, which can be checked in polynomial time. Alternatively, we can efficiently check whether each of the $|V|$ subsets of exactly $|V| - 1$ vertices is a vertex cover.

   Because the statement assumes that $\mathsf{P} \neq \mathsf{NP}$, this language is not NP-complete.

2. Suppose we independently flip 10000 *biased* coins, each resulting in a head with probability 1%. You want to bound the probability that at least 150 heads are flipped. Let X be the random variable representing the number of heads.

   Select the **tightest correct bound that can be obtained from either Markov's inequality or the Chernoff bound**, whichever applies.

   ◯ $\Pr[X \geq 150] \leq \frac{100}{150}$

   ◯ $\Pr[X \geq 150] \geq \frac{100}{150}$

   ● $\Pr[X \geq 150] \leq e^{-25/3}$

   ◯ $\Pr[X \geq 150] \leq e^{-50/3}$

   ◯ Neither Markov nor Chernoff applies to this setting.

   Solution: Because X is a sum of independent random variables between 0 and 1, we can apply the Chernoff bound. We have $\mathbb{E}[X] = 10000/100 = 100$ and $150 = (1 + 1/2)\,\mathbb{E}[X]$, so by the small-deviation (upper) Chernoff bound with $\lambda = 1/2$,

   $$\Pr[X \geq 150] = \Pr[X \geq (1 + 1/2)\,\mathbb{E}[X]] \leq \exp(-(1/2)^2\,\mathbb{E}[X]/3) = \exp(-25/3)\ .$$

   We cannot get a bound of $\exp(-50/3)$ using any of the Chernoff (or Chernoff-Hoeffding) bounds we have seen.

3. Alice wants to use the RSA system with modulus $n = 33 = 3 \cdot 11$. Select the **private exponent d and public exponent e that are a valid choice for this modulus**.

   ◯ $d = 3, e = 22$

   ◯ $d = 9, e = 10$

   ◯ $d = 17, e = 2$

   ● $d = 3, e = 7$

   ◯ $d = 2, e = 17$

Solution: Recall that RSA requires $e \times d \equiv 1 \pmod{(p-1)(q-1)}$. Here p, q are 11 and 3 (in either order), so $(11-1)(3-1) = 20$. The only pair whose product is congruent to 1 modulo 20 is $e = 3$ and $d = 7$.

4. Select the problem that is ***not known*** **to admit an efficient deterministic algorithm**.

   ◯ Given an integer n, check if n is odd.

   ◯ Given positive integers m, i, n, compute $m^i \bmod n$.

   ◯ Given integers m, n, compute $m \cdot n$.

   ◯ Given $n = pq$ for some distinct primes p, q and $\phi(n) = (p-1)(q-1)$, compute the set $\{p, q\}$.

   ● **Given positive integers m, t, n, compute an integer i such that $m^i \equiv t \pmod{n}$, if such an i exists.**

Solution: The first three problems (checking parity, integer multiplication, and modular exponentiation) have efficient algorithms, as shown in class. For the fourth one, we have that $n - \phi(n) + 1 = p + q$. Given that we know pq and $p + q$, we can efficiently solve for p and q. (This algorithm was shown in one of the discussion worksheets.)

The last problem is (a generalization of) the discrete log problem. No polynomial-time deterministic algorithm is known for it (although a polynomial-time *quantum* algorithm is known).

5. Select the value of $3^{3333} \bmod 11$.

   ◯ 1

   ◯ 3

   ● **5**

   ◯ 6

   ◯ 9

Solution: Fermat's Little Theorem says that for any prime p and any $a \in \mathbb{Z}_p^*$, we have that $a^{p-1} \equiv 1 \pmod{p}$. (Or, for any integers a and k, we have that $a^{1+k(p-1)} \equiv a \pmod{p}$.)

So, $3^{3330} \equiv 1 \pmod{11}$, and hence $3^{3333} \equiv 3^3 \equiv 27 \equiv 5 \pmod{11}$.

6. Let $G = (V, E)$ be a *directed* graph with no self-loop edge. For any subset $S \subseteq V$ of vertices, let

$$E(S) = \{(u, v) \in E : u \in S, v \notin S\}$$

   denote the set of edges going from S to outside S.

   Suppose that, for each vertex $v \in V$, we independently include v in S with probability 1/2.

   Select the **expected value of** $|E(S)|$.

   ◯ $|E|/2$

   ● $|E|/4$

   ◯ $|V|/2$

   ◯ $|V|/4$

   ◯ 0

Solution: Let X_e be the indicator whether a (directed) edge $e = (u, v)$ has $u \in S$ and $v \notin S$. Observe that by independence,

$$\mathbb{E}[X_e] = \Pr[u \in S] \cdot \Pr[v \notin S] = (1/2)^2 = 1/4.$$

Since $|E(S)| = \sum_{e \in E} X_e$, by linearity of expectation, $\mathbb{E}[|E(S)|] = \sum_{e \in E} \mathbb{E}[X_e] = |E|/4$.

UNIQNAME (print): _____

Multiple Choice: Select <u>all</u> valid options

For each of the problems in this section, select <u>all</u> valid options; this could be all of them, none of them, or something in between.

The scoring for each problem is as follows: 5 points for all five correct (non-)selections; and 3, 2, 1, 0 points for four, three, two, one (or zero) correct (non-)selections, respectively.

1. Select all of the following that are **known *not* to exist**.

   - ■ **A language that is efficiently decidable and not efficiently verifiable.**
   - ■ **A language that is undecidable and NP-Complete.**
   - ☐ A language that is undecidable and NP-Hard.
   - ☐ A language that is not in P but is in NP.
   - ☐ A language that is in P and is NP-complete.

   Solution:

   - Such a language does not exist because $\mathsf{P} \subseteq \mathsf{NP}$, i.e., every efficiently decidable language is also efficiently verifiable.
   - Such a language does not exist because any NP-complete language is in NP by definition, and any language in NP is decidable.
   - Such a language exists: the halting problem is undecidable and NP-hard.
   - It is not known whether such a language exists. If $\mathsf{P} \neq \mathsf{NP}$, then every NP-complete problem is in NP, but not in P. So we do not know how to rule out the existence of such a language.
   - It is not known whether such a language exists. If $\mathsf{P} = \mathsf{NP}$, then every (nontrivial) language in P is also NP-complete. So we do not know how to rule out the existence of such a language.

2. The *long*-path (decision) problem is: given an undirected unweighted graph G, vertices s and t, and a budget k, determine whether there exists a simple path in G from s to t having length *at least* k. (Recall that a simple path visits each vertex at most once.)

   The *short*-path (decision) problem is defined identically, but with "at most" in place of "at least."

   Select **all of the statements that are known to be true**.

   - ■ **The *long*-path problem is in NP.**
   - ■ **The *long*-path problem is NP-hard, because there is a polynomial-time mapping reduction from the Hamiltonian-path problem to it.**
   - ■ **There is a polynomial-time mapping reduction from the *short*-path problem to the *long*-path problem.**
   - ■ **If $\mathsf{P} = \mathsf{NP}$, then the *short*-path problem is NP-complete.**
   - ■ **If $\mathsf{P} \neq \mathsf{NP}$, then the *short*-path problem is not NP-complete.**

   Solution:

   - A verifier $V(G, C)$ for this problem accepts if and only if the certificate C represents a simple path in G from s to t of length at least k, which can be checked in polynomial time. So G is a "yes" instance iff there exists C where $V(G, C)$ accepts. Hence, the problem is in NP.

- A reduction from the Hamiltonian-path problem to the long-path problem is as follows: given an instance $X = (G, (s, t))$ of the Hamiltonian-path problem, output the long-path instance $X' = (G, (s, t), k = n - 1)$, where n is the number of vertices. Observe that X is a "yes" instance of the Hamiltonian-path problem if and only if X' is a "yes" instance of the long-path problem: any Hamiltonian path from s to t is a simple path of length exactly $n - 1$, and conversely, any simple s-to-t path of length $n - 1$ must visit every vertex exactly once, and hence is a Hamiltonian path from s to t. Also note that X' can be trivially constructed in linear time.
- The short-path problem is in P because we can compute single-source (or all-pairs) shortest paths in polynomial time using Bellman-Ford (or Floyd-Warshall). So there is a polynomial-time mapping reduction from it to any nontrivial decision problem.
- If P = NP, then every nontrivial decision problem in P is NP-complete; so in particular, the short-path problem is.
- If P ≠ NP, then every problem in P is *not* NP-complete, including the short-path problem in particular.

3. Suppose that A is a 2/3-approximation algorithm for the MaxClique problem, and let $G = (V, E)$ be the (undirected, unweighted) input graph. Select **all of the true statements**.

☐ $A(G)$ must output a clique in G of size at least $2|V|/3$.

■ $A(G)$ **must output a clique that has at least half as many vertices as a largest clique in G.**

■ $A(G)$ **must output a clique that has at least two-thirds as many vertices as a largest clique in G.**

☐ $A(G)$ cannot output a largest clique in G.

☐ $A(G)$ must output a larger clique than $A'(G)$ does, where A' is a 1/2-approximation algorithm for MaxClique.

Solution: Let OPT denote the size of a maximum clique in G. Let ALG denote the size of the clique output by $A(G)$. By hypothesis, $\text{ALG} \geq \frac{2}{3}\text{OPT}$.

- This is false. It could be that $\text{ALG} = \frac{2}{3}\text{OPT}$ and $\text{OPT} < |V|$, so $\text{ALG} < \frac{2}{3}|V|$.
- This is true, since $\text{ALG} \geq \frac{2}{3}\text{OPT} > \frac{1}{2}\text{OPT}$.
- This is true by the hypothesis on A.
- This is false. It could be that $\text{ALG} = \text{OPT}$; we only know that $\text{ALG} \geq \frac{2}{3}\text{OPT}$.
- This is false. Let ALG' denote the size of the clique output by $A'(G)$. We cannot conclude anything about how ALG and ALG' compare solely from the fact that $\text{ALG} \geq \frac{2}{3}\text{OPT}$ and $\text{ALG}' \geq \frac{1}{2}\text{OPT}$. Either one might be larger than the other, or they might be equal.

4. Select **all of the statements that are known to be true**.

■ **Suppose that processes A and B, which might depend on each other, fail with probabilities p_a and p_b, respectively. Then $\Pr[\text{both processes succeed}] \geq 1 - p_a - p_b$.**

☐ Let $X = \sum_{i=1}^{n} X_i$ for random variables $X_i \in [0, 1]$. Then $\Pr[X \geq 2\,\mathbb{E}[X]] \geq 1/2$.

- ☐ Let $X = \sum_{i=1}^{n} X_i$, where X_i is the indicator random variable for whether the ith U-Michigan student (out of a total of n) plays volleyball with their friends on the sand court outside Beyster, the day after finals are over. The Chernoff bound can be used to bound the probability that $X \geq 2\,\mathbb{E}[X]$.
- ☐ If you repeatedly toss a fair coin until you get a head, the expected number of coin tosses is infinite.
- ■ **If a program takes T steps in expectation, then it takes less than $4T$ steps with probability at least $3/4$.**

Solution:

- One of the process fails with probability $p_a + p_b$ by the union bound.
- Counterexample: let $n = 1$ and $X = 1$ with certainty. Then $X = \mathbb{E}[X] < 2\,\mathbb{E}[X]$ with certainty.
- The Chernoff bound requires independent random variables, but these X_i are dependent. There are many correlations among them: for example, friends (or enemies) are more (or less) likely to make the same decision about whether to play volleyball. If many people play volleyball and keep the court occupied, then others will be less likely to play.
- The expected number of coin tosses is 2.
- This is true because by Markov's inequality, it takes $\geq 4T$ steps with probability at most $1/4$.

5. Select **all of the statements that are known to be true.**
   - ☐ If $\mathsf{P} = \mathsf{NP}$, then every NP-hard language is in P.
   - ☐ If 3SAT is efficiently verifiable, then every language in NP is efficiently decidable.
   - ■ **There *exists* an NP-hard language in P if and only if *every* NP-complete language is in P.**
   - ■ **HamiltonianCycle $\in \mathsf{P}$ if and only if SubsetSum $\in \mathsf{P}$.**
   - ☐ Clique $\notin \mathsf{P}$.

Solution:

- This is false because L_{ACC} is NP-hard but undecidable (and therefore not in P), regardless of whether $\mathsf{P} = \mathsf{NP}$.
- 3SAT is known to be efficiently verifiable, but it is not known whether every language in NP is efficiently decidable; this is the P-versus-NP question.
- This is one of the central theorems of NP-completeness. If some NP-hard language is in P, then by reduction, every language in NP (and in particular, every NP-complete language) is in P. The other direction holds trivially.
- Both languages are NP-complete, so either both of them are in P, or neither is.
- Clique is NP-complete, so we do not know whether it is in P; this question is equivalent to the P-versus-NP question.

6. Select **all of the statements that are known to be true**.

- ☐ The one-time pad is information-theoretically secure even if the secret key is drawn from an arbitrary non-uniform distribution, as long as it is used only once.
- ■ **If there is an efficient algorithm for solving the discrete log problem, then there is an efficient algorithm for computing the shared secret key from the public messages in the Diffie-Hellman protocol.**
- ☐ If there is an efficient algorithm for computing the shared secret key from the public messages in the Diffie-Hellman protocol, then there is an efficient algorithm for solving the discrete log problem.
- ■ **For all integers m, we have $(m^{27})^3 \equiv m \pmod{55}$.**
- ■ **If a secure commitment scheme exists, then there is a zero-knowledge proof for 3SAT.**

Solution:

- The one-time pad is not necessarily secure if the secret key is not uniformly random. A counterexample is when all characters of the secret key are the same, as in the Caesar cipher. Another counterexample is if the key is a fixed string, without any random choice; then anyone can decrypt the ciphertext.
- This is directly from class. The eavesdropper can get Alice's and/or Bob's secret exponent by computing the discrete log of their public message, then use that to efficiently compute the shared secret key.
- This is directly from class. It is unknown whether efficiently computing the shared secret key *requires* solving the discrete log problem, or if there is some 'shortcut'.
- We have $55 = 5 \cdot 11$, so $\phi(55) = 4 \cdot 10 = 40$. By Euler's theorem, $m^{1+40k} \equiv m \pmod{55}$ for any integers m, k. Since $27 \cdot 3 = 81 = 1 + 40 \cdot 2$, we have that $(m^{27})^3 \equiv m^{81} \equiv m \pmod{55}$ for all integers m.
- This is directly from class. Assuming that a secure commitment scheme exists, there is a zero-knowledge proof for HamiltonianCycle, and hence for *every* NP problem (via poly-time mapping reduction).

UNIQNAME (print): _____

Short Answers — 7 Points Each

1. Alice and Bob agree on the prime $p = 19$ and generator $g = 10$ of $\mathbb{Z}_p^*$, and wish to establish a shared secret key using the Diffie–Hellman protocol.

   Based on their secret exponents, Alice sends $x = 5$ and Bob sends $y = 17$.

   Derive one of their secret exponents and their shared secret key, filling in the blanks in the sentence below. All values should be from $\mathbb{Z}_{19}^* = \{1, \ldots, 18\}$.

   The secret exponent of **Alice/Bob** (**write one name**) is **2 / 8**, and the shared secret key is **4**.

   Briefly justify (in 1–2 sentences) your answers.

   Solution: We have $10^2 \equiv 100 \equiv 5 \pmod{19}$, so Alice's exponent is 2. So, the shared secret key is $17^2 \equiv (-2)^2 \equiv 4 \pmod{19}$.

   Detailed Solution: We describe other ways of answering the problem below.

   Let a and b be the secret exponents of Alice and Bob, respectively. We have $10^a \equiv 5 \bmod 19$ and $10^b \equiv 17 \bmod 19$. We can find a and/or b by raising 10 to various powers, modulo 19:

   $$10^2 \equiv 100 \equiv 5 \pmod{19}$$
   $$10^4 \equiv 5^2 \equiv 25 \equiv 6 \pmod{19}$$
   $$10^8 \equiv 6^2 \equiv 36 \equiv 17 \pmod{19}$$

   Since 10 is a generator of $\mathbb{Z}_{19}^*$, the first equation implies that $a = 2$, and the third one implies that $b = 8$.

   By the Diffie-Hellman protocol, the shared secret key is

   $$17^a \equiv 5^b \equiv 10^{ab} \pmod{19}\ .$$

   Since $a = 2$ is small, it is most convenient to compute the first of these, and we can also use the fact that $17 \equiv -2 \pmod{19}$ to keep the numbers small:

   $$17^2 \equiv (-2)^2 \equiv 4 \pmod{19}\ .$$

   So, the shared secret key is 4.

   Alternatively, using $b = 8$ we could compute the shared key $5^8 \bmod 19$ using three repeated squarings:

   $$5^8 \equiv 25^4 \equiv 6^4 \equiv 36^2 \equiv 17^2 \equiv (-2)^2 \equiv 4 \pmod{19}\ .$$

   Or, using $ab = 16$ we could compute the shared key $10^{16} \bmod 19$ using four repeated squarings, which repeat the ones considered above.

2. Recall that a *triangle* in an undirected graph $G = (V, E)$ is a set $T = \{x, y, z\} \subseteq V$ of three (distinct) vertices where $(x, y), (y, z), (x, z) \in E$.

   A *triangle cover* in G is a subset of vertices $C \subseteq V$ such that *every* triangle T in G has at least one vertex in C, i.e., $T \cap C \neq \emptyset$.

   The *minimum triangle cover* problem MinTriCover is: given G, find a triangle cover of minimum size, i.e., having as few vertices as possible. It is known that the decision version of this problem is NP-complete.

   Consider the following (greedy, polynomial-time) algorithm for approximating MinTriCover.

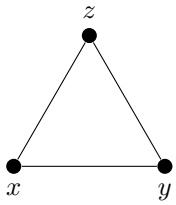
```
1: function GreedyTriCover(G)
2:     C ← ∅
3:     while there is some 'uncovered' triangle T in G where T ∩ C = ∅ do
4:         C ← C ∪ T, i.e., add all the vertices of T to C
5:     return C
```

   (a) **Correctly fill in the blank** in the following claim (you will justify your answer in the next parts). The ***smallest* approximation factor** $\alpha \geq 1$ that GreedyTriCover is *guaranteed* to obtain is

   $$\alpha = \underline{\quad \mathbf{3} \quad} .$$

   (b) **Draw a small input graph** for which the approximation factor α you gave in the previous part is *tight*, i.e., GreedyTriCover **necessarily obtains that factor, and no better**. Also, **briefly justify** (in 1-2 sentences) why this is so.

   Solution: The simplest example is a graph that consists of a single triangle:

   GreedyTriCover chooses $T = \{x, y, z\}$, includes all three vertices of T in the cover, and outputs it. An optimal triangle cover is any one vertex, so the obtained approximation factor is 3 (and no better).

   (c) **Briefly prove** (in 3-4 sentences) that GreedyTriCover obtains the approximation factor α you gave in the first part, on any input graph.

   Solution: (This is a slight adaptation of the proof for the DoubleCover approximation algorithm for VertexCover.)
   Let $T_1, \ldots, T_m$ be the sequence of triangles that GreedyTriCover(G) selects. By design, these triangles do not share any vertex. So, *any* triangle cover of G must have *at least* m vertices (because it must cover these triangles, at least), i.e., the optimal size $\text{OPT} \geq m$. Since the algorithm outputs a triangle cover with exactly $\text{ALG} = 3m$ vertices, it is guaranteed to obtain an approximation factor of $\text{ALG}/\text{OPT} \leq 3$.

3. An instance of the MaxMod3 problem is m equations involving n variables $x_1, \ldots, x_n \in \mathbb{Z}_3 = \{0, 1, 2\}$. The jth equation has the form

$$a_j + b_j \equiv c_j \pmod 3 ,$$

where a_j and b_j are ***distinct variables*** from $x_1, \ldots, x_n$, and $c_j \in \mathbb{Z}_3$ is a ***constant***. The goal is to find an assignment to the variables that maximizes the number of satisfied equations.

For example, if $x_1 = 1$, $x_2 = 2$, $x_3 = 0$, then the equation $x_1 + x_2 \equiv 0 \pmod 3$ is satisfied, but $x_1 + x_3 \equiv 2 \pmod 3$ is not satisfied.

Consider the algorithm that assigns each variable x_i a uniformly random and independent value in $\mathbb{Z}_3$.

(a) **Correctly fill in the blank**: the expected number of satisfied equations is _____$m/3$_____.

(b) **Rigorously justify** the answer you gave in the previous part using indicator random variables.

Solution: Let X_j be the indicator random variable for whether the jth equation is satisfied. We claim that

$$\mathbb{E}[X_j] = 1/3.$$

This is because for any fixed values of a_j and c_j, the probability that $b_j \equiv c_j - a_j \pmod 3$ is $1/3$, since b_j independent of a_j (for this it is important that they are *different* variables). So by linearity of expectation, the expected number of satisfied equations is

$$\mathbb{E}[\sum_{j=1}^{m} X_j] = \sum_{j=1}^{m} \mathbb{E}[X_j] = m/3.$$

A less nice solution: To see that $\mathbb{E}[X_j] = 1/3$, we can also list all possible outcomes of a_j and b_j. Then, we see that, for every $c_j \in \{0, 1, 2\}$, there are exactly 3 out of 9 outcomes where $a_j + b_j \equiv c_j \pmod 3$.

a_j	b_j	$a_j + b_j \bmod 3$
0	0	0
0	1	1
0	2	2
1	0	1
1	1	2
1	2	0
2	0	2
2	1	0
2	2	1

Long Answers

Question 1 is worth 14 points. Question 2 is worth 17 points.

1. Prove that the language

$$L_{\text{ExactFour}} = \{\langle M\rangle : M \text{ is a TM that accepts exactly four distinct inputs}\}$$

is undecidable, by reduction from either L_{ACC} or $L_{\varepsilon\text{-HALT}}$. Recall that their definitions are

$$L_{\text{ACC}} = \{(\langle M\rangle, x) : M \text{ is a Turing machine that accepts } x\}\ ,$$
$$L_{\varepsilon\text{-HALT}} = \{\langle M\rangle : M \text{ is a Turing machine that halts on } \varepsilon\}\ .$$

Solution: Since L_{ACC} is undecidable, it suffices to show $L_{\text{ACC}} \leq_T L_{\text{ExactFour}}$. Let $D_{\text{ExactFour}}$ be an oracle for deciding $L_{\text{ExactFour}}$. We construct a decider D_{ACC} for L_{ACC} as follows:

```
1: function D_ACC(⟨M⟩, x)
2:     S = {00, 01, 10, 11} (any four distinct strings will do)
3:     construct the following Turing machine M′:
4:     function M′(w)
5:         if w ∈ S then output M(x)
6:         reject (or loop forever)
7:     output D_ExactFour(⟨M′⟩)
```

Observe that D_{ACC} halts on all inputs since $D_{\text{ExactFour}}$ halts on all inputs. It remains to show that $(\langle M\rangle, x) \in L_{\text{ACC}} \iff D_{\text{ACC}}(\langle M\rangle, x)$ accepts. The key fact is that $M(x)$ accepts if and only if M' accepts exactly four distinct input strings, by the design of M'.

If M accepts x, then M' accepts exactly four inputs, namely, every $w \in S$. Therefore, $D_{\text{ExactFour}}(\langle M'\rangle)$ accepts, and so $D_{\text{ACC}}(\langle M\rangle, x)$ accepts.

If M does not accept x (i.e., it rejects or loops), then M' does not accept any input. So $D_{\text{ExactFour}}(\langle M'\rangle)$ rejects, hence $D_{\text{ACC}}(\langle M\rangle, x)$ rejects.

Another solution: Since $L_{\varepsilon\text{-HALT}}$ is undecidable, it suffices to show $L_{\varepsilon\text{-HALT}} \leq_T L_{\text{ExactFour}}$. Let $D_{\text{ExactFour}}$ be an oracle for deciding $L_{\text{ExactFour}}$. We construct a decider $D_{\varepsilon\text{-HALT}}$ for $L_{\varepsilon\text{-HALT}}$ as follows:

```
1: function D_ε-HALT(⟨M⟩)
2:     S = {00, 01, 10, 11} (any four distinct strings will do)
3:     construct the following Turing machine M′:
4:     function M′(w)
5:         if w ∈ S and M(ε) halts then accept
6:         reject (or loop forever)
7:     output D_ExactFour(⟨M′⟩)
```

Observe that $D_{\varepsilon\text{-HALT}}$ halts on all inputs since $D_{\text{ExactFour}}$ halts on all inputs. It remains to show that $\langle M\rangle \in L_{\varepsilon\text{-HALT}} \iff D_{\varepsilon\text{-HALT}}(\langle M\rangle)$ accepts. The key fact is that $M(\varepsilon)$ halts if and only if M' accepts exactly four distinct input strings, by the design of M'.

If $M(\varepsilon)$ halts, then M' accepts exactly four inputs, namely, every $w \in S$. Therefore, $D_{\text{ExactFour}}(\langle M'\rangle)$ accepts, and so $D_{\varepsilon\text{-HALT}}(\langle M\rangle)$ accepts.

If $M(\varepsilon)$ loops, then M' does not accept any input. So $D_{\text{ExactFour}}(\langle M'\rangle)$ rejects, hence $D_{\varepsilon\text{-HALT}}(\langle M\rangle)$ rejects.

2. Consider the following one-player (solitaire) game, played on a finite rectangular grid of cells—which may have any number of rows and columns—and with stones that are each colored *black* or *white*.

   Initially, each cell has zero or more stones, which may be of the same or different colors. The goal of the game is to remove zero or more of the stones from the grid, so that:

   1. each row has only one color of stone, or no stone at all, and
   2. each column has at least one stone.

   Some initial setups can be solved (and may even have multiple solutions), while others cannot be solved.

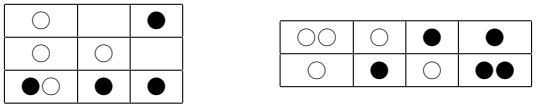


   Figure 1: In the above examples, ○ represents a white stone and ● represents a black stone. The first example is solvable by removing the bottom-left white stone and the top-right black stone. The second example cannot be solved.

   The SOLITAIRE decision problem is: given an n-by-m grid with an initial setup of stones, determine whether it can be solved.

   (a) **Briefly justify** (in 2–3 sentences, without pseudocode) why SOLITAIRE is in NP.

   > **Solution:** A certificate for a given instance of SOLITAIRE would just specify which stones to remove. A verifier can efficiently check whether the two conditions are met (after the specified stones have been removed). By definition, a given instance is solvable if and only if there is a certificate that causes the verifier to accept.
   >
   > Alternatively, a certificate could specify which stones to keep; the verifier would work similarly.

   (b) For showing that SOLITAIRE is NP-hard, **define a polynomial-time mapping reduction** f from 3SAT to SOLITAIRE, and **briefly justify its polynomial running time** (in 1–2 sentences). Do not address correctness; you will prove that in the next parts.

   Hint: Construct an initial setup where the rows correspond to the variables, the columns correspond to the clauses, and there are exactly three stones per column.

   > **Solution:** Given a 3SAT instance ϕ with n variables $x_1, \ldots, x_n$ and m clauses $c_1, \ldots, c_m$, create a grid $f(\phi)$ with n rows and m columns, and stones placed as follows:
   >
   > - If literal x_i appears in clause c_j, then put a white stone at entry (i, j).
   > - If literal $\neg x_i$ appears in clause c_j, then put a black stone at entry (i, j).
   >
   > (If multiple copies of the same literal appear in a clause, we can place multiple corresponding stones, or just one; either way is correct.)
   >
   > The reduction takes polynomial time in the size of the formula ϕ, because it allocates a grid of polynomial size nm, and it places (at most) one stone per literal in the formula.

(c) **Prove** that $\phi \in$ 3SAT $\implies f(\phi) \in$ Solitaire for your reduction f.

Solution: If $\phi \in$ 3SAT, there is a satisfying assignment for ϕ; fix some such assignment for the variables x_i. We show that there is a corresponding solution for the puzzle $f(\phi)$, i.e., $f(\phi) \in$ Solitaire. For each variable x_i:

- If $x_i =$ true, remove every black stone in row i of $f(\phi)$.
- If $x_i =$ false, remove every white stone in row i of $f(\phi)$.

Clearly, each row i has stones of only one color, or has no stone at all. It remains to show that each column j has at least one stone. This is because clause c_j has at least one true literal under the assignment, so the corresponding stone(s) will not have been removed.

More precisely, if clause c_j has true literal x_i, then the white stone in entry (i, j) is not removed. Similarly, if clause c_j has true literal $\neg x_i$, then the black stone in entry (i, j) is not removed.

(d) **Prove** that $f(\phi) \in$ Solitaire $\implies \phi \in$ 3SAT for your reduction f.

Solution: If $f(\phi) \in$ Solitaire, it has a solution. We construct a corresponding satisfying assignment for ϕ by setting the variables x_i as follows. Fix any solution to $f(\phi)$ and consider the resulting grid (where each row has only one color of stone, or no stone at all):

- If row i contains only white stones, set $x_i =$ true.
- If row i contains only black stones, set $x_i =$ false.
- If row i contains no stones, set x_i arbitrarily.

Since each column j has at least one stone, the clause c_j is satisfied, by the design of $f(\phi)$. Specifically, if there is a white stone at cell (i, j), then the literal x_i appears in c_j, and because we set $x_i =$ true, clause c_j is satisfied. Similar logic holds if there is a black stone in column j.

Because every clause is satisfied by our assignment, $\phi \in$ 3SAT, as desired.

This page is intentionally left blank for scratch work.

If you have answers on this page, write "answer continues on page 16" in the corresponding solution box.

This page is intentionally left blank for scratch work.

If you have answers on this page, write "answer continues on page 17" in the corresponding solution box.