The following problems ask you to prove some "obvious" claims about recursively-defined string functions. In each case, we want a self-contained, step-by-step induction proof that builds on formal definitions and prior reults, *not* on intuition. In particular, your proofs must refer to the formal recursive definitions of string length and string concatenation:

$$|w| := \begin{cases} 0 & \text{if } w = \varepsilon \\ 1 + |x| & \text{if } w = ax \text{ for some symbol } a \text{ and some string } x \end{cases}$$

$$w \bullet z := \begin{cases} z & \text{if } w = \varepsilon \\ a \cdot (x \bullet z) & \text{if } w = ax \text{ for some symbol } a \text{ and some string } x \end{cases}$$

You may freely use the following results, which are proved in the lecture notes:

Lemma 1: $w \bullet \varepsilon = w$ for all strings w.

Lemma 2: $|w \bullet x| = |w| + |x|$ for all strings w and x.

Lemma 3: $(w \bullet x) \bullet y = w \bullet (x \bullet y)$ for all strings w, x, and y.

The ***reversal*** $\boldsymbol{w^R}$ of a string w is defined recursively as follows:

$$w^R := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ x^R \bullet a & \text{if } w = ax \text{ for some symbol } a \text{ and some string } x \end{cases}$$

For example, $\texttt{STRESSED}^R = \texttt{DESSERTS}$ and $\texttt{WTF374}^R = \texttt{473FTW}$.

1. Prove that $|w| = |w^R|$ for every string w.

2. Prove that $(w \bullet z)^R = z^R \bullet w^R$ for all strings w and z.

3. Prove that $(w^R)^R = w$ for every string w.

[Hint: The proof for problem 3 relies on problem 2, but you may find it easier to solve problem 3 first.]

To think about later: Let $\#(a, w)$ denote the number of times symbol a appears in string w. For example, $\#(\texttt{X}, \texttt{WTF374}) = 0$ and $\#(\texttt{0}, \texttt{000010101010010100}) = 12$.

4. Give a formal recursive definition of $\#(a, w)$.

5. Prove that $\#(a, w \bullet z) = \#(a, w) + \#(a, z)$ for all symbols a and all strings w and z.

6. Prove that $\#(a, w^R) = \#(a, w)$ for all symbols a and all strings w.

Give regular expressions for each of the following languages over the binary alphabet {0, 1}.

1. All strings containing the substring 000.

2. All strings *not* containing the substring 000.

3. All strings in which every run of 0s has length at least 3.

4. All strings in which all the 1s appear before any substring 000.

5. All strings containing at least three 0s.

6. Every string except 000. *[Hint: Don't try to be clever.]*

Work on these later:

7. All strings w such that *in every prefix of w*, the number of 0s and 1s differ by at most 1.

⋆8. All strings containing at least two 0s and at least one 1.

⋆9. All strings w such that *in every prefix of w*, the number of 0s and 1s differ by at most 2.

★10. All strings in which the substring 000 appears an even number of times.
(For example, 0001000 and 0000 are in this language, but 00000 is not.)

Describe deterministic finite-state automata that accept each of the following languages over the alphabet $\Sigma = \{0, 1\}$. Describe briefly what each state in your DFAs *means*. Yes, these are exactly the same languages that you saw last Friday.

Either drawings or formal descriptions are acceptable, as long as the states Q, the start state s, the accept states A, and the transition function δ are all be clear. Try not to use too many states, but *don't* try to use as few states as possible.

1. All strings containing the substring 000.
2. All strings *not* containing the substring 000.
3. All strings in which every run of 0s has length at least 3.
4. All strings in which all the 1s appear before any substring 000.
5. All strings containing at least three 0s.
6. Every string except 000. *[Hint: Don't try to be clever.]*

Work on these later:

7. All strings w such that *in every prefix of w*, the number of 0s and 1s differ by at most 1.
8. All strings containing at least two 0s and at least one 1.
9. All strings w such that *in every prefix of w*, the number of 0s and 1s differ by at most 2.

*10. All strings in which the substring 000 appears an even number of times.
(For example, 0001000 and 0000 are in this language, but 00000 is not.)

Describe deterministic finite-state automata that accept each of the following languages over the alphabet $\Sigma = \{0, 1\}$. You may find it easier to describe these DFAs formally than to draw pictures.

Either drawings or formal descriptions are acceptable, as long as the states Q, the start state s, the accept states A, and the transition function δ are all clear. Try to keep the number of states small.

1. All strings in which the number of 0s is even **and** the number of 1s is *not* divisible by 3.

2. All strings in which the number of 0s is even **or** the number of 1s is *not* divisible by 3.

3. All strings that are **both** the binary representation of an integer divisible by 3 **and** the ternary (base-3) representation of an integer divisible by 4.

   For example, the string 1100 is an element of this language, because it represents $2^3 + 2^2 = 12$ in binary and $3^3 + 3^2 = 36$ in ternary.

Work on these later:

3. All strings in which the subsequence 0101 appears an even number of times.

4. All strings w such that $\binom{|w|}{2} \bmod 6 = 4$.
   [Hint: Maintain both $\binom{|w|}{2} \bmod 6$ *and* $|w| \bmod 6$.*]*
   [Hint: $\binom{n+1}{2} = \binom{n}{2} + n$.*]*

*5. All strings w such that $F_{\#(10,w)} \bmod 10 = 4$, where $\#(10, w)$ denotes the number of times 10 appears as a substring of w, and F_n is the nth Fibonacci number:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

Prove that each of the following languages is ***not*** regular.

1. $\left\{\mathtt{0}^{2^n} \;\middle|\; n \geq 0\right\}$

2. $\{\mathtt{0}^{2n}\mathtt{1}^n \mid n \geq 0\}$

3. $\{\mathtt{0}^m\mathtt{1}^n \mid m \neq 2n\}$

4. Strings over $\{\mathtt{0}, \mathtt{1}\}$ where the number of `0`s is exactly twice the number of `1`s.

5. Strings of properly nested parentheses `()`, brackets `[]`, and braces `{}`. For example, the string `([]){}` is in this language, but the string `([)]` is not, because the left and right delimiters don't match.

Work on these later:

6. Strings of the form $w_1\mathtt{\#}w_2\mathtt{\#}\cdots\mathtt{\#}w_n$ for some $n \geq 2$, where each substring w_i is a string in $\{\mathtt{0}, \mathtt{1}\}^*$, and some pair of substrings w_i and w_j are equal.

7. $\left\{\mathtt{0}^{n^2} \;\middle|\; n \geq 0\right\}$

8. $\{w \in (\mathtt{0} + \mathtt{1})^* \mid w \text{ is the binary representation of a perfect square}\}$

1. (a) Convert the regular expression $(\mathtt{0}^*\mathtt{1} + \mathtt{0}\mathtt{1}^*)^*$ into an NFA using Thompson's algorithm.

   (b) Convert the NFA you just constructed into a DFA using the incremental subset construction. Draw the resulting DFA. Your DFA should have four states, all reachable from the start state. (Some of these states are obviously equivalent, but keep them separate.)

   (c) **Think about later:** Convert the DFA you constructed in part (b) into a regular expression using Han and Wood's algorithm. You should *not* get the same regular expression you started with.

   (d) **Think about later:** Find the smallest DFA that is equivalent to your DFA from part (b) and convert that smaller DFA into a regular expression using Han and Wood's algorithm. Again, you should *not* get the same regular expression you started with.

   (e) What is this language?

2. (a) Convert the regular expression $(\varepsilon + (\mathtt{0} + \mathtt{11})^*\mathtt{0})\mathtt{1}(\mathtt{11})^*$ into an NFA using Thompson's algorithm.

   (b) Convert the NFA you just constructed into a DFA using the incremental subset construction. Draw the resulting DFA. Your DFA should have six states, all reachable from the start state. (Some of these states are obviously equivalent, but keep them separate.)

   (c) **Think about later:** Convert the DFA you constructed in part (b) into a regular expression using Han and Wood's algorithm. You should *not* get the same regular expression you started with.

   (d) **Think about later:** Find the smallest DFA that is equivalent to your DFA from part (b), using Moore's algorithm (in Section 3.10 of the notes), and convert that smaller DFA into a regular expression using Han and Wood's algorithm. Again, you should *not* get the same regular expression you started with.

   (e) What is this language?

Consider the following recursively defined function on strings:

$$stutter(w) := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ aa \bullet stutter(x) & \text{if } w = ax \text{ for some symbol } a \text{ and some string } x \end{cases}$$

Intuitively, $stutter(w)$ doubles every symbol in w. For example:

- $stutter(\texttt{PRESTO}) = \texttt{PPRREESSTTOO}$
- $stutter(\texttt{HOCUS}\diamond\texttt{POCUS}) = \texttt{HHOOCCUUSS}\diamond\diamond\texttt{PPOOCCUUSS}$

Let L be an arbitrary regular language.

1. Prove that the language $stutter^{-1}(L) := \{w \mid stutter(w) \in L\}$ is regular.
2. Prove that the language $stutter(L) := \{stutter(w) \mid w \in L\}$ is regular.

Work on these later:

3. Let L be an arbitrary regular language.

   (a) Prove that the language $insert1(L) := \{x1y \mid xy \in L\}$ is regular.
   Intuitively, $insert1(L)$ is the set of all strings that can be obtained from strings in L by inserting exactly one $\texttt{1}$. For example, if $L = \{\varepsilon, \texttt{OOK!}\}$, then $insert1(L) = \{\texttt{1}, \texttt{1OOK!}, \texttt{O1OK!}, \texttt{OO1K!}, \texttt{OOK1!}, \texttt{OOK!1}\}$.

   (b) Prove that the language $delete1(L) := \{xy \mid x1y \in L\}$ is regular.
   Intuitively, $delete1(L)$ is the set of all strings that can be obtained from strings in L by deleting exactly one $\texttt{1}$. For example, if $L = \{\texttt{101101}, \texttt{00}, \varepsilon\}$, then $delete1(L) = \{\texttt{01101}, \texttt{10101}, \texttt{10110}\}$.

4. Consider the following recursively defined function on strings:

$$evens(w) := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ \varepsilon & \text{if } w = a \text{ for some symbol } a \\ b \cdot evens(x) & \text{if } w = abx \text{ for some symbols } a \text{ and } b \text{ and some string } x \end{cases}$$

   Intuitively, $evens(w)$ skips over every other symbol in w. For example:

   - $evens(\texttt{EXPELLIARMUS}) = \texttt{XELAMS}$
   - $evens(\texttt{AVADA}\diamond\texttt{KEDAVRA}) = \texttt{VD}\diamond\texttt{EAR}$.

   Once again, let L be an arbitrary regular language.

   (a) Prove that the language $evens^{-1}(L) := \{w \mid evens(w) \in L\}$ is regular.

   (b) Prove that the language $evens(L) := \{evens(w) \mid w \in L\}$ is regular.

You saw the following context-free grammars in class on Thursday; in each example, the grammar itself is on the left; the explanation for each non-terminal is on the right.

- Properly nested strings of parentheses.

  $$S \to \varepsilon \mid S(S) \qquad\qquad \text{properly nested parentheses}$$

  Here is a different grammar for the same language:

  $$S \to \varepsilon \mid (S) \mid SS \qquad\qquad \text{properly nested parentheses}$$

- $\{0^m1^n \mid m \neq n\}$. This is the set of all binary strings composed of some number of 0s followed by a *different* number of 1s.

  $$\begin{aligned} S &\to A \mid B && \{0^m1^n \mid m \neq n\}\\ A &\to 0A \mid 0C && \{0^m1^n \mid m > n\}\\ B &\to B1 \mid C1 && \{0^m1^n \mid m < n\}\\ C &\to \varepsilon \mid 0C1 && \{0^m1^n \mid m = n\} \end{aligned}$$

Give context-free grammars for each of the following languages. For each grammar, describe the language for each non-terminal, either in English or using mathematical notation, as in the examples above. We probably won't finish all of these during the lab session.

1. $\{0^{2n}1^n \mid n \geq 0\}$

2. $\{0^m1^n \mid m \neq 2n\}$

   [Hint: If $m \neq 2n$, then either $m < 2n$ or $m > 2n$. Extend the previous grammar, but pay attention to parity. This language contains the string 01*.]*

3. $\{0,1\}^* \setminus \{0^{2n}1^n \mid n \geq 0\}$

   [Hint: Extend the previous grammar. What's missing?]

Work on these later:

4. $\big\{w \in \{0,1\}^* \bigm| \#(0,w) = 2 \cdot \#(1,w)\big\}$ — Binary strings where the number of 0s is exactly twice the number of 1s.

*5. $\{0,1\}^* \setminus \{ww \mid w \in \{0,1\}^*\}$.

   [Anti-hint: The language $\{ww \mid w \in 0,1^*\}$ is **not** *context-free. Thus, the complement of a context-free language is not necessarily context-free!]*

Let L be an arbitrary regular language over the alphabet $\Sigma = \{\mathtt{0}, \mathtt{1}\}$. Prove that the following languages are also regular. (You probably won't get to all of these.)

1. $\textsc{FlipOdds}(L) := \{\mathit{flipOdds}(w) \mid w \in L\}$, where the function *flipOdds* inverts every odd-indexed bit in w. For example:

$$\mathit{flipOdds}(\mathtt{0000111101010101}) = \mathtt{1010010111111111}$$

Solution: Let $M = (Q, s, A, \delta)$ be a DFA that accepts L. We construct a new DFA $M' = (Q', s', A', \delta')$ that accepts $\textsc{FlipOdds}(L)$ as follows.

Intuitively, M' receives some string $\mathit{flipOdds}(w)$ as input, restores every other bit to obtain w, and simulates M on the restored string w.

Each state (q, flip) of M' indicates that M is in state q, and we need to flip the next input bit if $\mathit{flip} = \textsc{True}$

$$\begin{aligned} Q' &= Q \times \{\textsc{True}, \textsc{False}\} \\ s' &= (s, \textsc{True}) \\ A' &= \\ \delta'((q, \mathit{flip}), a) &= \end{aligned}$$

■

2. $\textsc{UnflipOdd}\mathtt{1}\textsc{s}(L) := \{w \in \Sigma^* \mid \mathit{flipOdd1s}(w) \in L\}$, where the function *flipOdd1* inverts every other $\mathtt{1}$ bit of its input string, starting with the first $\mathtt{1}$. For example:

$$\mathit{flipOdd1s}(\mathtt{0000\underline{1}1\underline{1}10\underline{1}010\underline{1}01}) = \mathtt{0000\underline{0}1\underline{0}100\underline{0}010\underline{0}01}$$

Solution: Let $M = (Q, s, A, \delta)$ be a DFA that accepts L. We construct a new DFA $M' = (Q', s', A', \delta')$ that accepts $\textsc{UnflipOdd}\mathtt{1}\textsc{s}(L)$ as follows.

Intuitively, M' receives some string w as input, flips every other $\mathtt{1}$ bit, and simulates M on the transformed string.

Each state (q, flip) of M' indicates that M is in state q, and we need to flip the next $\mathtt{1}$ bit of and only if $\mathit{flip} = \textsc{True}$.

$$\begin{aligned} Q' &= Q \times \{\textsc{True}, \textsc{False}\} \\ s' &= (s, \textsc{True}) \\ A' &= \\ \delta'((q, \mathit{flip}), a) &= \end{aligned}$$

■

3. FlipOdd1s$(L) := \{\mathit{flipOdd1s}(w) \mid w \in L\}$, where the function *flipOdd1* is defined as in the previous problem.

Solution: Let $M = (Q, s, A, \delta)$ be a DFA that accepts L. We construct a new **NFA** $M' = (Q', s', A', \delta')$ that accepts FlipOdd1s(L) as follows.

Intuitively, M' receives some string $\mathit{flipOdd1s}(w)$ as input, ***guesses*** which 0 bits to restore to 1s, and simulates M on the restored string w. No string in FlipOdd1s(L) has two 1s in a row, so if M' ever sees 11, it rejects.

Each state (q, flip) of M' indicates that M is in state q, and we need to flip a 0 bit before the next 1 if $\mathit{flip} = \text{True}$.

$$
\begin{aligned}
Q' &= Q \times \{\text{True}, \text{False}\} \\
s' &= (s, \text{True}) \\
A' &= \\
\delta'((q, \mathit{flip}), a) &=
\end{aligned}
$$

■

4. Faro$(L) := \left\{\mathit{faro}(w, x) \;\middle|\; w, x \in L \text{ and } |w| = |x|\right\}$, where the function *faro* is defined recursively as follows:

$$
\mathit{faro}(w, x) := \begin{cases} x & \text{if } w = \varepsilon \\ a \cdot \mathit{faro}(x, y) & \text{if } w = ay \text{ for some } a \in \Sigma \text{ and some } y \in \Sigma^* \end{cases}
$$

For example, $\mathit{faro}(0001101, 1111001) = 01010111100011$. (A "faro shuffle" splits a deck of cards into two equal piles and then perfectly interleaves them.)

Solution: Let $M = (Q, s, A, \delta)$ be a DFA that accepts L. We construct a DFA $M' = (Q', s', A', \delta')$ that accepts Faro(L) as follows.

Intuitively, M' reads the string $\mathit{faro}(w, x)$ as input, splits the string into the subsequences w and x, and passes each of those strings to an independent copy of M.

Each state $(q_1, q_2, \mathit{next})$ indicates that the copy of M that gets w is in state q_1, the copy of M that gets x is in state q_2, and *next* indicates which copy gets the next input bit.

$$
\begin{aligned}
Q' &= Q \times Q \times \{1, 2\} \\
s' &= (s, s, 1) \\
A' &= \\
\delta'((q_1, q_2, \mathit{next}), a) &=
\end{aligned}
$$

■

Here are several problems that are easy to solve in $O(n)$ time, essentially by brute force. Your task is to design algorithms for these problems that are significantly faster.

1. Suppose we are given an array $A[1\,..\,n]$ of n distinct integers, which could be positive, negative, or zero, sorted in increasing order so that $A[1] < A[2] < \cdots < A[n]$.

   (a) Describe a fast algorithm that either computes an index i such that $A[i] = i$ or correctly reports that no such index exists.

   (b) Suppose we know in advance that $A[1] > 0$. Describe an even faster algorithm that either computes an index i such that $A[i] = i$ or correctly reports that no such index exists. *[Hint: This is **really** easy.]*

2. Suppose we are given an array $A[1\,..\,n]$ such that $A[1] \geq A[2]$ and $A[n-1] \leq A[n]$. We say that an element $A[x]$ is a ***local minimum*** if both $A[x-1] \geq A[x]$ and $A[x] \leq A[x+1]$. For example, there are exactly six local minima in the following array:

9	7	7	2	1	3	7	5	4	7	3	3	4	8	6	9
	▲			▲				▲		▲	▲			▲	

   Describe and analyze a fast algorithm that returns the index of one local minimum. For example, given the array above, your algorithm could return the integer 9, because $A[9]$ is a local minimum. *[Hint: With the given boundary conditions, any array **must** contain at least one local minimum. Why?]*

3. Suppose you are given two sorted arrays $A[1\,..\,n]$ and $B[1\,..\,n]$ containing distinct integers. Describe a fast algorithm to find the median (meaning the nth smallest element) of the union $A \cup B$. For example, given the input

$$A[1\,..\,8] = [0, 1, 6, 9, 12, 13, 18, 20] \qquad B[1\,..\,8] = [2, 4, 5, 8, 17, 19, 21, 23]$$

   your algorithm should return the integer 9. *[Hint: What can you learn by comparing one element of A with one element of B?]*

To think about later:

4. Now suppose you are given two sorted arrays $A[1\,..\,m]$ and $B[1\,..\,n]$ and an integer k. Describe a fast algorithm to find the kth smallest element in the union $A \cup B$. For example, given the input

$$A[1\,..\,8] = [0, 1, 6, 9, 12, 13, 18, 20] \qquad B[1\,..\,5] = [2, 5, 7, 17, 19] \qquad k = 6$$

   your algorithm should return the integer 7.

In lecture, Jeff described an algorithm of Karatsuba that multiplies two n-digit integers using $O(n^{\lg 3})$ single-digit additions, subtractions, and multiplications. In this lab we'll look at some extensions and applications of this algorithm.

1. Describe an algorithm to compute the product of an n-digit number and an m-digit number, where $m < n$, in $O(m^{\lg 3 - 1} n)$ time.

2. Describe an algorithm to compute the decimal representation of 2^n in $O(n^{\lg 3})$ time.

   [Hint: Repeated squaring. The standard algorithm that computes one decimal digit at a time requires $\Theta(n^2)$ time.]

3. Describe a divide-and-conquer algorithm to compute the decimal representation of an arbitrary n-bit binary number in $O(n^{\lg 3})$ time.

   [Hint: Let $x = a \cdot 2^{n/2} + b$. Watch out for an extra log factor in the running time.]

Think about later:

4. Suppose we can multiply two n-digit numbers in $O(M(n))$ time. Describe an algorithm to compute the decimal representation of an arbitrary n-bit binary number in $O(M(n) \log n)$ time.

A ***subsequence*** of a sequence (for example, an array, linked list, or string), obtained by removing zero or more elements and keeping the rest in the same sequence order. A subsequence is called a ***substring*** if its elements are contiguous in the original sequence. For example:

- `SUBSEQUENCE`, `UBSEQU`, and the empty string ε are all substrings (and therefore subsequences) of the string `SUBSEQUENCE`;
- `SBSQNC`, `SQUEE`, and `EEE` are all subsequences of `SUBSEQUENCE` but not substrings;
- `QUEUE`, `EQUUS`, and `DIMAGGIO` are not subsequences (and therefore not substrings) of `SUBSEQUENCE`.

Describe ***recursive backtracking*** algorithms for the following longest-subsequence problems. *Don't worry about running times.*

1. Given an array $A[1\,..\,n]$ of integers, compute the length of a longest ***increasing*** subsequence. A sequence $B[1\,..\,\ell]$ is *increasing* if $B[i] > B[i-1]$ for every index $i \geq 2$.

   For example, given the array

   $$\left\langle 3, \underline{1}, \underline{4}, 1, \underline{5}, 9, 2, \underline{6}, 5, 3, 5, \underline{8}, \underline{9}, 7, 9, 3, 2, 3, 8, 4, 6, 2, 7 \right\rangle$$

   your algorithm should return the integer 6, because $\langle 1, 4, 5, 6, 8, 9 \rangle$ is a longest increasing subsequence (one of many).

2. Given an array $A[1\,..\,n]$ of integers, compute the length of a longest ***decreasing*** subsequence. A sequence $B[1\,..\,\ell]$ is *decreasing* if $B[i] < B[i-1]$ for every index $i \geq 2$.

   For example, given the array

   $$\left\langle 3, 1, 4, 1, 5, \underline{9}, 2, \underline{6}, 5, 3, \underline{5}, 8, 9, 7, 9, 3, 2, 3, 8, \underline{4}, 6, \underline{2}, 7 \right\rangle$$

   your algorithm should return the integer 5, because $\langle 9, 6, 5, 4, 2 \rangle$ is a longest decreasing subsequence (one of many).

3. Given an array $A[1\,..\,n]$ of integers, compute the length of a longest ***alternating*** subsequence. A sequence $B[1\,..\,\ell]$ is *alternating* if $B[i] < B[i-1]$ for every even index $i \geq 2$, and $B[i] > B[i-1]$ for every odd index $i \geq 3$.

   For example, given the array

   $$\left\langle \underline{3}, \underline{1}, \underline{4}, \underline{1}, \underline{5}, 9, \underline{2}, \underline{6}, \underline{5}, 3, 5, \underline{8}, 9, \underline{7}, \underline{9}, \underline{3}, 2, 3, \underline{8}, \underline{4}, \underline{6}, \underline{2}, \underline{7} \right\rangle$$

   your algorithm should return the integer 17, because $\langle 3, 1, 4, 1, 5, 2, 6, 5, 8, 7, 9, 3, 8, 4, 6, 2, 7 \rangle$ is a longest alternating subsequence (one of many).

To think about later:

4. Given an array $A[1\,..\,n]$ of integers, compute the length of a longest ***convex*** subsequence of A. A sequence $B[1\,..\,\ell]$ is *convex* if $B[i] - B[i-1] > B[i-1] - B[i-2]$ for every index $i \geq 3$.

   For example, given the array

   $$\langle \underline{\mathbf{3}}, \underline{\mathbf{1}}, 4, \underline{\mathbf{1}}, 5, 9, \underline{\mathbf{2}}, 6, 5, 3, \underline{\mathbf{5}}, 8, \underline{\mathbf{9}}, 7, 9, 3, 2, 3, 8, 4, 6, 2, 7 \rangle$$

   your algorithm should return the integer 6, because $\langle 3, 1, 1, 2, 5, 9 \rangle$ is a longest convex subsequence (one of many).

5. Given an array $A[1\,..\,n]$, compute the length of a longest ***palindrome*** subsequence of A. Recall that a sequence $B[1\,..\,\ell]$ is a *palindrome* if $B[i] = B[\ell - i + 1]$ for every index i.

   For example, given the array

   $$\langle 3, 1, \underline{\mathbf{4}}, 1, 5, \underline{\mathbf{9}}, 2, 6, \underline{\mathbf{5}}, \underline{\mathbf{3}}, \underline{\mathbf{5}}, 8, 9, 7, \underline{\mathbf{9}}, 3, 2, 3, 8, \underline{\mathbf{4}}, 6, 2, 7 \rangle$$

   your algorithm should return the integer 7, because $\langle 4, 9, 5, 3, 5, 9, 4 \rangle$ is a longest palindrome subsequence (one of many).

A ***subsequence*** of a sequence (for example, an array, a linked list, or a string), obtained by removing zero or more elements and keeping the rest in the same sequence order. A subsequence is called a ***substring*** if its elements are contiguous in the original sequence. For example:

- SUBSEQUENCE, UBSEQU, and the empty string ε are all substrings of the string SUBSEQUENCE;
- SBSQNC, UEQUE, and EEE are all subsequences of SUBSEQUENCE but not substrings;
- QUEUE, SSS, and FOOBAR are not subsequences of SUBSEQUENCE.

Describe and analyze ***dynamic programming*** algorithms for the following longest-subsequence problems. Use the recursive backtracking algorithms you developed on Wednesday.

1. Given an array $A[1..n]$ of integers, compute the length of a longest ***increasing*** subsequence of A. A sequence $B[1..\ell]$ is *increasing* if $B[i] > B[i-1]$ for every index $i \geq 2$.

2. Given an array $A[1..n]$ of integers, compute the length of a longest ***decreasing*** subsequence of A. A sequence $B[1..\ell]$ is *decreasing* if $B[i] < B[i-1]$ for every index $i \geq 2$.

3. Given an array $A[1..n]$ of integers, compute the length of a longest ***alternating*** subsequence of A. A sequence $B[1..\ell]$ is *alternating* if $B[i] < B[i-1]$ for every even index $i \geq 2$, and $B[i] > B[i-1]$ for every odd index $i \geq 3$.

4. Given an array $A[1..n]$ of integers, compute the length of a longest ***convex*** subsequence of A. A sequence $B[1..\ell]$ is *convex* if $B[i]-B[i-1] > B[i-1]-B[i-2]$ for every index $i \geq 3$.

5. Given an array $A[1..n]$, compute the length of a longest ***palindrome*** subsequence of A. Recall that a sequence $B[1..\ell]$ is a *palindrome* if $B[i] = B[\ell-i+1]$ for every index i.

Basic steps in developing a dynamic programming algorithm

1. **Formulate the problem recursively.** This is the hard part. There are two distinct but equally important things to include in your formulation.

    (a) **Specification.** First, give a clear and precise English description of the problem you are claiming to solve. Not *how* to solve the problem, but *what* the problem actually is. Omitting this step in homeworks or exams is an automatic zero.

    (b) **Solution.** Second, give a clear recursive formula or algorithm for the whole problem in terms of the answers to smaller instances of *exactly* the same problem. It generally helps to think in terms of a recursive definition of your inputs and outputs. If you discover that you need a solution to a *similar* problem, or a slightly *related* problem, you're attacking the wrong problem; go back to step 1.

2. **Build solutions to your recurrence from the bottom up.** Write an algorithm that starts with the base cases of your recurrence and works its way up to the final solution, by considering intermediate subproblems in the correct order. This stage can be broken down into several smaller, relatively mechanical steps:

    (a) **Identify the subproblems.** What are all the different ways can your recursive algorithm call itself, starting with some initial input?

    (b) **Analyze running time.** Add up the running times of all possible subproblems, *ignoring the recursive calls*.

    (c) **Choose a memoization data structure.** For most problems, each recursive subproblem can be identified by a few integers, so you can use a multidimensional array. But some problems need a more complicated data structure.

    (d) **Identify dependencies.** Except for the base cases, every recursive subproblem depends on other subproblems—which ones? Draw a picture of your data structure, pick a generic element, and draw arrows from each of the other elements it depends on. Then formalize your picture.

    (e) **Find a good evaluation order.** Order the subproblems so that each subproblem comes *after* the subproblems it depends on. Typically, you should consider the base cases first, then the subproblems that depends only on base cases, and so on. ***Be careful!***

    (f) **Write down the algorithm.** You know what order to consider the subproblems, and you know how to solve each subproblem. So do that! If your data structure is an array, this usually means writing a few nested for-loops around your original recurrence.

Nancy Gunter, the founding dean of the new Maksymilian R. Levchin College of Computing, has commissioned a series of snow ramps on the south slope of the Orchard Downs sledding hill[1] and challenged Erhan Hajek, head of the Department of Electrical and Computer Engineering, to a sledding contest. Erhan and Nancy will both sled down the hill, each trying to maximize their air time. The winner gets to expand their department/college into Siebel Center, the ECE Building, *and* the new Campus Instructional Facility; the loser has to move their entire department/college under the Boneyard bridge behind Everitt Lab.

Whenever Nancy or Erhan reaches a ramp *while on the ground*, they can either use that ramp to jump through the air, possibly flying over one or more ramps, or sled past that ramp and stay on the ground. Obviously, if someone flies over a ramp, they cannot use that ramp to extend their jump.

1. Suppose you are given a pair of arrays $Ramp[1..n]$ and $Length[1..n]$, where $Ramp[i]$ is the distance from the top of the hill to the ith ramp, and $Length[i]$ is the distance that any sledder who takes the ith ramp will travel through the air.

   Describe and analyze an algorithm to determine the maximum *total* distance that Erhan and Nancy can travel through the air.

2. Uh-oh. The university lawyers heard about Nancy and Erhan's little bet and immediately objected. To protect the university from both lawsuits and sky-rocketing insurance rates, they impose an upper bound on the number of jumps that either sledder can take.

   Describe and analyze an algorithm to determine the maximum total distance that Nancy or Erhan can spend in the air *with at most k jumps*, given the original arrays $Ramp[1..n]$ and $Length[1..n]$ and the integer k as input.

3. **To think about later:** When the lawyers realized that imposing their restriction didn't immediately shut down the contest, they added yet another restriction: No ramp may be used more than once! Disgusted by all the legal interference, Erhan and Nancy give up on their bet and decide to cooperate to put on a good show for the spectators.

   Describe and analyze an algorithm to determine the maximum total distance that Nancy and Erhan can spend in the air, each taking at most k jumps (so at most $2k$ jumps total), and with each ramp used at most once.

[1] The north slope is faster, but too short for an interesting contest.

1. A ***basic arithmetic expression*** is composed of characters from the set $\{1, +, \times\}$ and parentheses. Almost every integer can be represented by more than one basic arithmetic expression. For example, all of the following basic arithmetic expression represent the integer 14:

$$1+1+1+1+1+1+1+1+1+1+1+1+1+1$$
$$((1+1)\times(1+1+1+1+1))+((1+1)\times(1+1))$$
$$(1+1)\times(1+1+1+1+1+1+1)$$
$$(1+1)\times(((1+1+1)\times(1+1))+1)$$

Describe and analyze an algorithm to compute, given an integer n as input, the minimum number of 1's in a basic arithmetic expression whose value is equal to n. The number of parentheses doesn't matter, just the number of 1's. For example, when $n = 14$, your algorithm should return 8, for the final expression above. The running time of your algorithm should be bounded by a small polynomial function of n.

Think about later:

2. Suppose you are given a sequence of integers separated by + and − signs; for example:

$$1+3-2-5+1-6+7$$

You can change the value of this expression by adding parentheses in different places. For example:

$$1+3-2-5+1-6+7=-1$$
$$(1+3-(2-5))+(1-6)+7=9$$
$$(1+(3-2))-(5+1)-(6+7)=-17$$

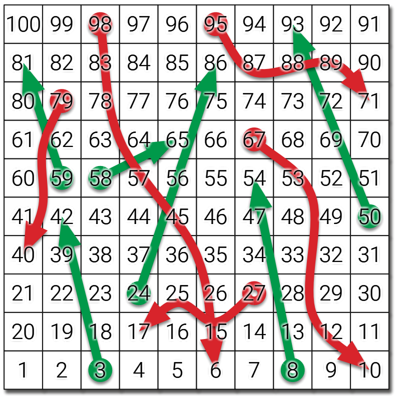
Describe and analyze an algorithm to compute, given a list of integers separated by + and − signs, the maximum possible value the expression can take by adding parentheses. Parentheses must be used only to group additions and subtractions; in particular, do not use them to create implicit multiplication as in $1+3(-2)(-5)+1-6+7=33$.

For each of the problems below, transform the input into a graph and apply a standard graph algorithm that you've seen in class. Whenever you use a standard graph algorithm, you ***must*** provide the following information. (I recommend actually using a bulleted list.)

- What are the vertices? What does each vertex represent?
- What are the edges? Are they directed or undirected?
- If the vertices and/or edges have associated values, what are they?
- What problem do you need to solve on this graph?
- What standard algorithm are you using to solve that problem?
- What is the running time of your entire algorithm, *including* the time to build the graph, *as a function of the original input parameters*?

---

1. ***Snakes and Ladders*** is a classic board game, originating in India no later than the 16th century. The board consists of an $n \times n$ grid of squares, numbered consecutively from 1 to n^2, starting in the bottom left corner and proceeding row by row from bottom to top, with rows alternating to the left and right. Certain pairs of squares, always in different rows, are connected by either "snakes" (leading down) or "ladders" (leading up). ***Each square can be an endpoint of at most one snake or ladder.***



100	99	98	97	96	95	94	93	92	91
81	82	83	84	85	86	87	88	89	90
80	79	78	77	76	75	74	73	72	71
61	62	63	64	65	66	67	68	69	70
60	59	58	57	56	55	54	53	52	51
41	42	43	44	45	46	47	48	49	50
40	39	38	37	36	35	34	33	32	31
21	22	23	24	25	26	27	28	29	30
20	19	18	17	16	15	14	13	12	11
1	2	3	4	5	6	7	8	9	10

A typical Snakes and Ladders board.
Upward straight arrows are ladders; downward wavy arrows are snakes.

You start with a token in cell 1, in the bottom left corner. In each move, you advance your token up to k positions, for some fixed constant k (typically 6). Then if the token is at the *top* of a snake, you ***must*** slide the token down to the bottom of that snake, and if the token is at the *bottom* of a ladder, you ***may*** move the token up to the top of that ladder.

Describe and analyze an efficient algorithm to compute the smallest number of moves required for the token to reach the last square of the Snakes and Ladders board.

2. Let G be an undirected graph. Suppose we start with two coins on two arbitrarily chosen vertices of G. At every step, each coin ***must*** move to an adjacent vertex. Describe and analyze an efficient algorithm to compute the minimum number of steps to reach a configuration where both coins are on the same vertex, or to report correctly that no such configuration is reachable. The input to your algorithm consists of a graph $G = (V, E)$ and two vertices $u, v \in V$ (which may or may not be distinct).

Think about later:

3. Let G be an undirected graph. Suppose we start with 374 coins on 374 arbitrarily chosen vertices of G. At every step, each coin ***must*** move to an adjacent vertex. Describe and analyze an efficient algorithm to compute the minimum number of steps to reach a configuration where all 374 coins are on the same vertex, or to report correctly that no such configuration is reachable. The input to your algorithm consists of a graph $G = (V, E)$ and starting vertices $s_1, s_2, \ldots, s_{374}$ (which may or may not be distinct).

For each of the problems below, transform the input into a graph and apply a standard graph algorithm that you've seen in class. Whenever you use a standard graph algorithm, you ***must*** provide the following information. (I recommend actually using a bulleted list.)

- What are the vertices? What does each vertex represent?
- What are the edges? Are they directed or undirected?
- If the vertices and/or edges have associated values, what are they?
- What problem do you need to solve on this graph?
- What standard algorithm are you using to solve that problem?
- What is the running time of your entire algorithm, *including* the time to build the graph, *as a function of the original input parameters*?

---

1. Inspired by the previous lab, you decide to organize a Snakes and Ladders competition with n participants. In this competition, each game of Snakes and Ladders involves three players. After the game is finished, they are ranked first, second, and third. Each player may be involved in any (non-negative) number of games, and the number need not be equal among players.

   At the end of the competition, m games have been played. You realize that you forgot to implement a proper rating system, and therefore decide to produce the overall ranking of all n players as you see fit. However, to avoid being too suspicious, if player A ranked better than player B in any game, then A must rank better than B in the overall ranking.

   You are given the list of players and their ranking in each of the m games. Describe and analyze an algorithm that produces an overall ranking of the n players that is consistent with the individual game rankings, or correctly reports that no such ranking exists.

2. There are n galaxies connected by m intergalactic teleport-ways. Each teleport-way joins two galaxies and can be traversed in both directions. However, the company that runs the teleport-ways has established an extremely lucrative cost structure: Anyone can teleport *further* from their home galaxy at no cost whatsoever, but teleporting *toward* their home galaxy is prohibitively expensive.

   Judy has decided to take a sabbatical tour of the universe by visiting as many galaxies as possible, starting at her home galaxy. To save on travel expenses, she wants to teleport away from her home galaxy at every step, except for the very last teleport home.

   Describe and analyze an algorithm to compute the maximum number of galaxies that Judy can visit. Your input consists of an undirected graph G with n vertices and m edges describing the teleport-way network, an integer $1 \le s \le n$ identifying Judy's home galaxy, and an array $D[1..n]$ containing the distances of each galaxy from s.

To think about later:

3. Just before embarking on her universal tour, Judy wins the space lottery, giving her just enough money to afford *two* teleports toward her home galaxy. Describe and analyze a new algorithm to compute the maximum number of galaxies Judy can visit; if she visits the same galaxy twice, that counts as two visits. After all, argues the travel agent, who can see an entire galaxy in just one visit?

*4. Judy replies angrily to the travel agent that *she* can see an entire galaxy in just one visit, because 99% of every galaxy is exactly the same glowing balls of plasma and lifeless chunks of rock and McDonalds and Starbucks and prefab “Irish” pubs and overpriced souvenir shops and Peruvian street-corner musicians as every other galaxy.

Describe and analyze an algorithm to compute the maximum number of *distinct* galaxies Judy can visit. She is still *allowed* to visit the same galaxy more than once, but only the first visit counts toward her total.

1. Describe and analyze an algorithm to compute the shortest path from vertex s to vertex t in a directed graph with weighted edges, where exactly *one* edge $u{\to}v$ has negative weight. Assume the graph has no negative cycles. *[Hint: Modify the input graph and run Dijkstra's algorithm. Alternatively,* ***don't*** *modify the input graph, but run Dijkstra's algorithm anyway.]*

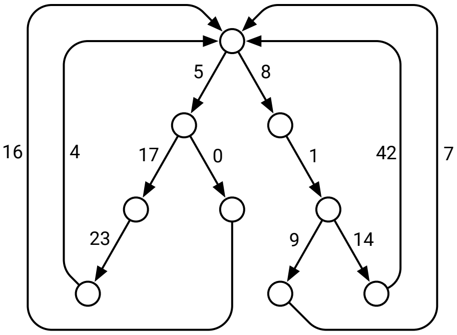
2. You just discovered your best friend from elementary school on Twitbook. You both want to meet as soon as possible, but you live in two different cities that are far apart. To minimize travel time, you agree to meet at an intermediate city, and then you simultaneously hop in your cars and start driving toward each other. But where *exactly* should you meet?

   You are given a weighted graph $G = (V, E)$, where the vertices V represent cities and the edges E represent roads that directly connect cities. Each edge e has a weight $w(e)$ equal to the time required to travel between the two cities. You are also given a vertex p, representing your starting location, and a vertex q, representing your friend's starting location.

   Describe and analyze an algorithm to find the target vertex t that allows you and your friend to meet as soon as possible, assuming both of you leave home *right now*.

To think about later:

3. A *looped tree* is a weighted, directed graph built from a binary tree by adding an edge from every leaf back to the root. Every edge has a non-negative weight.



A looped tree.

   (a) How much time would Dijkstra's algorithm require to compute the shortest path between two vertices u and v in a looped tree with n nodes?

   (b) Describe and analyze a faster algorithm.

1. Suppose that you have just finished computing the array $dist[1..V, 1..V]$ of shortest-path distances between ***all*** pairs of vertices in an edge-weighted directed graph G. Unfortunately, you discover that you incorrectly entered the weight of a single edge $u{\to}v$, so all that precious CPU time was wasted. Or was it? Maybe your distances are correct after all!

   In each of the following problems, let $w(u{\to}v)$ denote the weight that you used in your distance computation, and let $w'(u{\to}v)$ denote the correct weight of $u{\to}v$.

   (a) Suppose $w(u{\to}v) > w'(u{\to}v)$; that is, the weight you used for $u{\to}v$ was *larger* than its true weight. Describe an algorithm that repairs the distance array in $\boldsymbol{O(V^2)}$ ***time*** under this assumption. *[Hint: For every pair of vertices x and y, either $u{\to}v$ is on the shortest path from x to y or it isn't.]*

   (b) Maybe even that was too much work. Describe an algorithm that determines whether your original distance array is actually correct in $\boldsymbol{O(1)}$ ***time***, again assuming that $w(u{\to}v) > w'(u{\to}v)$. *[Hint: Either $u{\to}v$ is the shortest path from u to v or it isn't.]*

   (c) **To think about later:** Describe an algorithm that determines in $\boldsymbol{O(VE)}$ ***time*** whether your distance array is actually correct, even if $w(u{\to}v) < w'(u{\to}v)$.

   (d) **To think about later:** Argue that when $w(u{\to}v) < w'(u{\to}v)$, repairing the distance array *requires* recomputing shortest paths from scratch, at least in the worst case.

2. You—yes, *you*—can cause a major economic collapse with the power of graph algorithms![1] The *arbitrage* business is a money-making scheme that takes advantage of differences in currency exchange. In particular, suppose that 1 US dollar buys 120 Japanese yen; 1 yen buys 0.01 euros; and 1 euro buys 1.2 US dollars. Then, a trader starting with \$1 can convert their money from dollars to yen, then from yen to euros, and finally from euros back to dollars, ending with \$1.44! The cycle of currencies \$ → ¥ → € → \$ is called an ***arbitrage cycle***. Of course, finding and exploiting arbitrage cycles before the prices are corrected requires extremely fast algorithms.

   Suppose n different currencies are traded in your currency market. You are given the matrix $R[1..n]$ of exchange rates between every pair of currencies; for each i and j, one unit of currency i can be traded for $R[i, j]$ units of currency j. (Do *not* assume that $R[i, j] \cdot R[j, i] = 1$.)

   (a) Describe an algorithm that returns an array $V[1..n]$, where $V[i]$ is the maximum amount of currency i that you can obtain by trading, starting with one unit of currency 1, assuming there are no arbitrage cycles.

   (b) Describe an algorithm to determine whether the given matrix of currency exchange rates creates an arbitrage cycle.

   *(c) **To think about later:** Modify your algorithm from part (b) to actually return an arbitrage cycle, if such a cycle exists.

[1] No, you can't.

1. ***Flappy Bird*** is a once-popular mobile game written by Nguyễn Hà Đông, originally released in May 2013.[1] The game features a bird named "Faby", who flies to the right at constant speed. Whenever the player taps the screen, Faby is given a fixed upward velocity; between taps, Faby falls due to gravity. Faby flies through a landscape of pipes until it touches either a pipe or the ground, at which point the game is over. Your task, should you choose to accept it, is to develop an algorithm to play Flappy Bird automatically.

   Well, okay, not Flappy Bird exactly, but the following drastically simplified variant, which I will call ***Flappy Pixel***. Instead of a bird, Faby is a single point, specified by three integers: horizontal position x (in pixels), vertical position y (in pixels), and vertical speed y' (in pixels per frame). Faby's environment is described by two arrays $Hi[1\,..\,n]$ and $Lo[1\,..\,n]$, where for each index i, we have $0 < Lo[i] < Hi[i] < h$ for some fixed height value h. The game is described by the following piece of pseudocode:

```
FLAPPYPIXEL($Hi[1\,..\,n], Lo[1\,..\,n]$):
    $y \leftarrow \lceil h/2 \rceil$
    $y' \leftarrow 0$
    for $x \leftarrow 1$ to $n$
        if the player taps the screen
            $y' \leftarrow 10$          ⟨⟨flap⟩⟩
        else
            $y' \leftarrow y' - 1$      ⟨⟨fall⟩⟩
        $y \leftarrow y + y'$
        if $y < Lo[x]$ or $y > Hi[x]$
            return FALSE         ⟨⟨player loses⟩⟩
    return TRUE                  ⟨⟨player wins⟩⟩
```

   Notice that in each iteration of the main loop, the player has the option of tapping the screen.

   Describe and analyze an algorithm to determine the minimum number of times that the player must tap the screen to win Flappy Pixel, given the integer h and the arrays $Hi[1\,..\,n]$ and $Lo[1\,..\,n]$ as input. If the game cannot be won at all, your algorithm should return ∞. Describe the running time of your algorithm as a function of n and h.

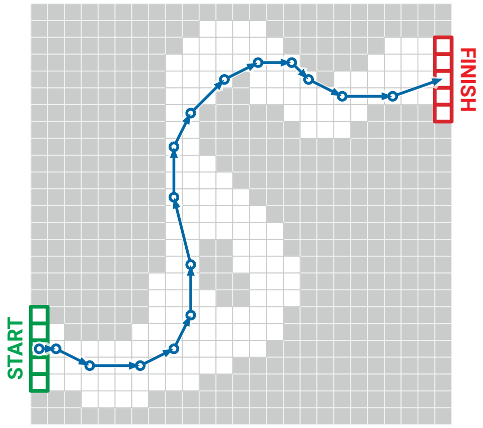
[Problem 2 is on the back.]

[1] In 2015, Đông pulled all official versions of Flappy Bird from all online app stores, expressing guilt for the addictive behavior his game had encouraged. Unfortunately, dozens of unofficial clones had already been lunched by then, so Đông's altruism was rendered moot. Although Flappy Bird has significantly waned in popularity as a mobile, you can still find officially licensed arcade versions that give you tickets which you can exchange for prizes that cost significantly less than the money you spent to win the tickets.

2. ***Racetrack*** (also known as *Graph Racers* and *Vector Rally*) is a two-player paper-and-pencil racing game that Jeff played on the bus in 5th grade.[2] The game is played with a track drawn on a sheet of graph paper. The players alternately choose a sequence of grid points that represent the motion of a car around the track, subject to certain constraints explained below.

   Each car has a *position* and a *velocity*, both with integer x- and y-coordinates. A subset of grid squares is marked as the *starting area*, and another subset is marked as the *finishing area*. The initial position of each car is chosen by the player somewhere in the starting area; the initial velocity of each car is always $(0,0)$. At each step, the player optionally changes each component of the velocity by at most 1. The car's new position is then determined by adding the new velocity to the car's previous position. The new position must be inside the track; otherwise, the car crashes and that player loses the race.[3] The race ends when the first car reaches a position inside the finishing area.

velocity	position
$(0,0)$	$(1,5)$
$(1,0)$	$(2,5)$
$(2,-1)$	$(4,4)$
$(3,0)$	$(7,4)$
$(2,1)$	$(9,5)$
$(1,2)$	$(10,7)$
$(0,3)$	$(10,10)$
$(-1,4)$	$(9,14)$
$(0,3)$	$(9,17)$
$(1,2)$	$(10,19)$
$(2,2)$	$(12,21)$
$(2,1)$	$(14,22)$
$(2,0)$	$(16,22)$
$(1,-1)$	$(17,21)$
$(2,-1)$	$(19,20)$
$(3,0)$	$(22,20)$
$(3,1)$	$(25,21)$



A 16-step Racetrack run, on a 25 × 25 track. This is *not* the shortest run on this track.

   Suppose the racetrack is represented by an $n \times n$ array of bits, where each `0` bit represents a grid point inside the track, each `1` bit represents a grid point outside the track, the "starting line" consists of all `0` bits in column 1, and the "finishing line" consists of all `0` bits in column n.

   Describe and analyze an algorithm to find the minimum number of steps required to move a car from the starting line to the finish line of a given racetrack.

   [Hint: Your initial analysis can be improved.]

[2] The actual game is a bit more complicated than the version described here. See http://harmmade.com/vectorracer/ for an excellent online version.

[3] However, it is not necessary for the entire line segment between the old position and the new position to lie inside the track. Sometimes Speed Racer has to push the A button.

To think about later:

3. Consider the following variant of Flappy Pixel. The mechanics of the game are unchanged, but now the environment is specified by an array $Points[1..n, 1..h]$ of integers, which could be positive, negative, or zero. If Faby falls off the top or bottom edge of the environment, the game immediately ends and the player gets nothing. Otherwise, at each frame, the player earns $Points[x, y]$ points, where (x, y) is Faby's current position. The game ends when Faby reaches the right end of the environment.

```
FlappyPixel2($Points[1..n]$):
    $score \gets 0$
    $y \gets \lceil h/2 \rceil$
    $y' \gets 0$
    for $x \gets 1$ to $n$
        if the player taps the screen
            $y' \gets 10$              ⟨⟨flap⟩⟩
        else
            $y' \gets y' - 1$          ⟨⟨fall⟩⟩
        $y \gets y + y'$
        if $y < 1$ or $y > h$
            return $-\infty$           ⟨⟨fail⟩⟩
        $score \gets score + Points[x, y]$
    return $score$
```

Describe and analyze an algorithm to determine the maximum possible score that a player can earn in this game.

4. We can also consider a similar variant of Racetrack. Instead of bits, the "track" is described by an array $Points[1..n, 1..n]$ of *numbers*, which could be positive, negative, or zero. Whenever the car lands on a grid cell (i, j), the player receives $Points[i, j]$ points. Forbidden grid cells are indicated by $Points[i, j] = -\infty$.

   Describe and analyze an algorithm to find the largest possible score that a player can earn by moving a car from column 1 (the starting line) to column n (the finish line).

   [Hint: Wait, what if all the point values are positive?]

1. Suppose you are given a magic black box that somehow answers the following decision problem in *polynomial time*:

   - INPUT: A boolean circuit K with n inputs and one output.
   - OUTPUT: TRUE if there are input values $x_1, x_2, \ldots, x_n \in \{\text{TRUE}, \text{FALSE}\}$ that make K output TRUE, and FALSE otherwise.

   Using this black box as a subroutine, describe an algorithm that solves the following related search problem *in polynomial time*:

   - INPUT: A boolean circuit K with n inputs and one output.
   - OUTPUT: Input values $x_1, x_2, \ldots, x_n \in \{\text{TRUE}, \text{FALSE}\}$ that make K output TRUE, or NONE if there are no such inputs.

   [Hint: You can use the magic box more than once.]

2. An ***independent set*** in a graph G is a subset S of the vertices of G, such that no two vertices in S are connected by an edge in G. Suppose you are given a magic black box that somehow answers the following decision problem *in polynomial time*:

   - INPUT: An undirected graph G and an integer k.
   - OUTPUT: TRUE if G has an independent set of size k, and FALSE otherwise.

   (a) Using this black box as a subroutine, describe algorithms that solves the following optimization problem *in polynomial time*:

   - INPUT: An undirected graph G.
   - OUTPUT: The size of the largest independent set in G.

   [Hint: You've seen this problem before.]

   (b) Using this black box as a subroutine, describe algorithms that solves the following search problem *in polynomial time*:

   - INPUT: An undirected graph G.
   - OUTPUT: An independent set in G of maximum size.

To think about later:

3. Formally, a ***proper coloring*** of a graph $G = (V, E)$ is a function $c \colon V \to \{1, 2, \ldots, k\}$, for some integer k, such that $c(u) \neq c(v)$ for all $uv \in E$. Less formally, a valid coloring assigns each vertex of G a color, such that every edge in G has endpoints with different colors. The ***chromatic number*** of a graph is the minimum number of colors in a proper coloring of G.

   Suppose you are given a magic black box that somehow answers the following decision problem *in polynomial time*:

   - INPUT: An undirected graph G and an integer k.
   - OUTPUT: TRUE if G has a proper coloring with k colors, and FALSE otherwise.

   Using this black box as a subroutine, describe an algorithm that solves the following ***coloring problem*** *in polynomial time*:

   - INPUT: An undirected graph G.
   - OUTPUT: A valid coloring of G using the minimum possible number of colors.

   *[Hint: You can use the magic box more than once. The input to the magic box is a graph and **only** a graph, meaning **only** vertices and edges.]*

Proving that a problem X is NP-hard requires several steps:

- Choose a problem Y that you already know is NP-hard (because we told you so in class).
- Describe an algorithm to solve Y, using an algorithm for X as a subroutine. Typically this algorithm has the following form: Given an instance of Y, transform it into an instance of X, and then call the magic black-box algorithm for X.
- ***Prove*** that your algorithm is correct. This always requires two separate steps, which are usually of the following form:
  - ***Prove*** that your algorithm transforms "good" instances of Y into "good" instances of X.
  - ***Prove*** that your algorithm transforms "bad" instances of Y into "bad" instances of X. Equivalently: Prove that if your transformation produces a "good" instance of X, then it was given a "good" instance of Y.
- Argue that your algorithm for Y runs in polynomial time. (This is usually trivial.)

1. Recall the following kColor problem: Given an undirected graph G, can its vertices be colored with k colors, so that every edge touches vertices with two different colors?

   (a) Describe a direct polynomial-time reduction from 3Color to 4Color.

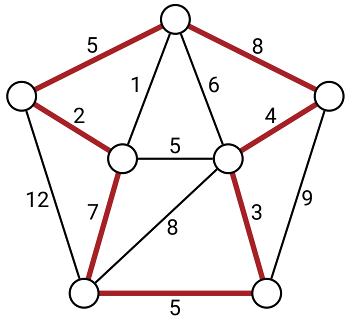
   (b) Prove that kColor problem is NP-hard for any $k \geq 3$.

2. A *Hamiltonian cycle* in a graph G is a cycle that goes through every vertex of G exactly once. Deciding whether an arbitrary graph contains a Hamiltonian cycle is NP-hard.

   A ***tonian cycle*** in a graph G is a cycle that goes through at least *half* of the vertices of G. Prove that deciding whether a graph contains a tonian cycle is NP-hard.

To think about later:

3. Let G be an undirected graph with weighted edges. A Hamiltonian cycle in G is ***heavy*** if the total weight of edges in the cycle is at least half of the total weight of all edges in G. Prove that deciding whether a graph contains a heavy Hamiltonian cycle is NP-hard.



A heavy Hamiltonian cycle. The cycle has total weight 34; the graph has total weight 67.

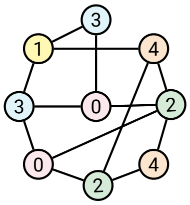
Prove that each of the following problems is NP-hard.

1. Given an undirected graph G, does G contain a simple path that visits all but 374 vertices?

2. Given an undirected graph G, does G have a spanning tree in which every node has degree at most 374?

3. Given an undirected graph G, does G have a spanning tree with at most 374 leaves?

1. Recall that a 5-coloring of a graph G is a function that assigns each vertex of G a “color” from the set $\{0,1,2,3,4\}$, such that for any edge uv, vertices u and v are assigned different “colors”. A 5-coloring is ***careful*** if the colors assigned to adjacent vertices are not only distinct, but differ by more than 1 (mod 5). Prove that deciding whether a given graph has a careful 5-coloring is NP-hard. *[Hint: Reduce from the standard 5Color problem.]*



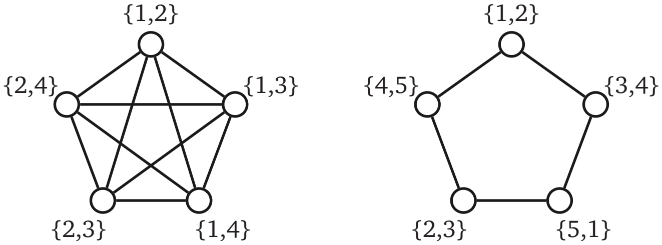
A careful 5-coloring.

2. Prove that the following problem is NP-hard: Given an undirected graph G, find *any* integer $k > 374$ such that G has a proper coloring with k colors but G does not have a proper coloring with $k - 374$ colors.

3. A ***bicoloring*** of an undirected graph assigns each vertex a set of *two* colors. There are two types of bicoloring: In a *weak* bicoloring, the endpoints of each edge must use *different* sets of colors; however, these two sets may share one color. In a *strong* bicoloring, the endpoints of each edge must use *distinct* sets of colors; that is, they must use four colors altogether. Every strong bicoloring is also a weak bicoloring.

   (a) Prove that finding the minimum number of colors in a weak bicoloring of a given graph is NP-hard.

   (b) Prove that finding the minimum number of colors in a strong bicoloring of a given graph is NP-hard.



Left: A weak bicoloring of a 5-clique with four colors.
Right A strong bicoloring of a 5-cycle with five colors.

Proving that a language L is undecidable by reduction requires several steps. (These are the essentially the same steps you already use to prove that a problem is NP-hard.)

- Choose a language L' that you already know is undecidable (because we told you so in class). The simplest choice is usually the standard halting language

  $$\textsc{Halt} := \big\{\langle M, w\rangle \;\big|\; M \text{ halts on } w\big\}$$

- Describe an algorithm that decides L', using an algorithm that decides L as a black box. Typically your reduction will have the following form:

  Given an arbitrary string x, construct a special string y,
  such that $y \in L$ if and only if $x \in L'$.

  In particular, if $L = \textsc{Halt}$, your reduction will have the following form:

  Given the encoding $\langle M, w\rangle$ of a Turing machine M and a string w,
  construct a special string y, such that
  $y \in L$ if and only if M halts on input w.

- Prove that your algorithm is correct. This proof almost always requires two separate steps:
  - Prove that if $x \in L'$ then $y \in L$.
  - Prove that if $x \notin L'$ then $y \notin L$.

Very important: Name every object in your proof, and *always* refer to objects by their names. Never *ever* refer to "the Turing machine" or "the algorithm" or "the code" or "the input string" or (gods forbid) "it" or "this", even in casual conversation, even if you're "just" explaining your intuition, even when you're "just" *thinking* about the reduction to yourself.

Prove that the following languages are undecidable.

1. $\textsc{AcceptIllini} := \big\{\langle M\rangle \;\big|\; M \text{ accepts the string } \texttt{ILLINI}\big\}$
2. $\textsc{AcceptThree} := \big\{\langle M\rangle \;\big|\; M \text{ accepts exactly three strings}\big\}$
3. $\textsc{AcceptPalindrome} := \big\{\langle M\rangle \;\big|\; M \text{ accepts at least one palindrome}\big\}$
4. $\textsc{AcceptOnlyPalindromes} := \big\{\langle M\rangle \;\big|\; \text{Every string accepted by } M \text{ is a palindrome}\big\}$

A solution for problem 1 appears on the next page; don't look at it until you've thought a bit about the problem first.

Solution (for problem 1): For the sake of argument, suppose there is an algorithm DecideAcceptIllini that correctly decides the language AcceptIllini. Then we can solve the halting problem as follows:

```
DecideHalt($\langle M, w\rangle$):
    Encode the following Turing machine $M'$:
            $M'(x)$:
                run $M$ on input $w$
                return True
    if DecideAcceptIllini($\langle M'\rangle$)
            return True
    else
            return False
```

We prove this reduction correct as follows:

$\Longrightarrow$ Suppose M halts on input w.

Then M' accepts *every* input string x.

In particular, M' accepts the string `ILLINI`.

So DecideAcceptIllini accepts the encoding $\langle M'\rangle$.

So DecideHalt correctly accepts the encoding $\langle M, w\rangle$.

$\Longleftarrow$ Suppose M does not halt on input w.

Then M' diverges on *every* input string x.

In particular, M' does not accept the string `ILLINI`.

So DecideAcceptIllini rejects the encoding $\langle M'\rangle$.

So DecideHalt correctly rejects the encoding $\langle M, w\rangle$.

In both cases, DecideHalt is correct. But that's impossible, because Halt is undecidable. We conclude that the algorithm DecideAcceptIllini does not exist. ■

As usual for undecidablility proofs, this proof invokes *four* distinct Turing machines:

- The hypothetical algorithm DecideAcceptIllini.
- The new algorithm DecideHalt that we construct in the solution.
- The arbitrary machine M whose encoding is part of the input to DecideHalt.
- The special machine M' whose encoding DecideHalt constructs (from the encoding of M and w) and then passes to DecideAcceptIllini.

Rice's Theorem. *Let $\mathcal{L}$ be any set of languages that satisfies the following conditions:*

- *There is a Turing machine Y such that $\textsc{Accept}(Y) \in \mathcal{L}$.*
- *There is a Turing machine N such that $\textsc{Accept}(N) \notin \mathcal{L}$.*

The language $\textsc{AcceptIn}(\mathcal{L}) := \big\{\langle M\rangle \bigm| \textsc{Accept}(M) \in \mathcal{L}\big\}$ is undecidable.

Prove that the following languages are undecidable *using Rice's Theorem*:

1. $\textsc{AcceptRegular} := \big\{\langle M\rangle \bigm| \textsc{Accept}(M) \text{ is regular}\big\}$
2. $\textsc{AcceptIllini} := \big\{\langle M\rangle \bigm| M \text{ accepts the string } \texttt{ILLINI}\big\}$
3. $\textsc{AcceptPalindrome} := \big\{\langle M\rangle \bigm| M \text{ accepts at least one palindrome}\big\}$
4. $\textsc{AcceptThree} := \big\{\langle M\rangle \bigm| M \text{ accepts exactly three strings}\big\}$
5. $\textsc{AcceptUndecidable} := \big\{\langle M\rangle \bigm| \textsc{Accept}(M) \text{ is undecidable } \big\}$

To think about later. Which of the following are undecidable? How would you prove that?

1. $\textsc{Accept}\{\{\varepsilon\}\} := \big\{\langle M\rangle \bigm| M \text{ accepts only the string } \varepsilon\text{; that is, } \textsc{Accept}(M) = \{\varepsilon\}\big\}$
2. $\textsc{Accept}\{\varnothing\} := \big\{\langle M\rangle \bigm| M \text{ does not accept any strings; that is, } \textsc{Accept}(M) = \varnothing\big\}$
3. $\textsc{Accept}\varnothing := \big\{\langle M\rangle \bigm| \textsc{Accept}(M) \text{ is not an acceptable language}\big\}$
4. $\textsc{Accept=Reject} := \big\{\langle M\rangle \bigm| \textsc{Accept}(M) = \textsc{Reject}(M) \ \big\}$
5. $\textsc{Accept}{\neq}\textsc{Reject} := \big\{\langle M\rangle \bigm| \textsc{Accept}(M) \neq \textsc{Reject}(M) \ \big\}$
6. $\textsc{Accept}{\cup}\textsc{Reject} := \big\{\langle M\rangle \bigm| \textsc{Accept}(M) \cup \textsc{Reject}(M) = \Sigma^*\big\}$

CS/ECE 374 A ✧ Fall 2019

Homework 0

Due Tuesday, September 3, 2019 at 8pm

- **Each student must submit individual solutions for this homework.** For all future homeworks, groups of up to three students can submit joint solutions.
- **Submit your solutions electronically to Gradescope as PDF files.** Submit a separate PDF file for each numbered problem. If you plan to typeset your solutions, please use the LaTeX solution template on the course web site. If you must submit scanned handwritten solutions, please use a black pen on blank white paper and a high-quality scanner app (or an actual scanner).
- You are ***not*** required to sign up on Gradescope or Piazza with your real name and your illinois.edu email address; you may use any email address and alias of your choice. However, to give you credit for the homework, we need to know who Gradescope thinks you are. **Please fill out the web form linked from the course web page.**

Some important course policies

- **You may use any source at your disposal**—paper, electronic, or human—but you ***must*** cite ***every*** source that you use, and you ***must*** write everything yourself in your own words. See the academic integrity policies on the course web site for more details.
- The answer ***"I don't know"*** (and *nothing* else) is worth 25% partial credit on any required problem or subproblem on any homework or exam. We will accept synonyms like "No idea" or "WTF" or "\(ó_ò)/", but you must write *something*.

  On the other hand, only the homework problems you submit actually contribute to your overall course grade, so submitting "I don't know" for an entire numbered homework problem will almost certainly hurt your grade more than submitting nothing at all.
- **Avoid the Three Deadly Sins!** Any homework or exam solution that breaks any of the following rules will be given an ***automatic zero***, unless the solution is otherwise perfect. Yes, we really mean it. We're not trying to be scary or petty (Honest!), but we do want to break a few common bad habits that seriously impede mastery of the course material.
  - Always give complete solutions, not just examples.
  - Always declare all your variables, in English. In particular, always describe the precise problem your algorithm is supposed to solve.
  - Never use weak induction.

See the course web site for more information.

If you have any questions about these policies,
please don't hesitate to ask in class, in office hours, or on Piazza.

1. The infamous Scottish computational arborist Seòras na Coille has a favorite 26-node binary tree, whose nodes are labeled with the letters of the English alphabet. Preorder and inorder traversals of his tree yield the following letter sequences:

   Preorder: U X M Z I W J E O V N H R D T K G L Y A F S Q P C B
   Inorder: Z M X E J V O W I N D R T H U G K F A Q P S Y C B L

   (a) List the nodes in Professor na Coille's tree according to a postorder traversal.

   (b) Draw Professor na Coille's tree.

   You do *not* need to prove that your answers are correct. *[Hint: It may be easier to write a short Python program than to figure this out by hand.]*

2. For any string $w \in \{0,1\}^*$, let $sort(w)$ denote the string obtained by sorting the characters in w. For example, $sort(010101) = 000111$. The $sort$ function can be defined recursively as follows:

$$sort(w) := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ 0 \cdot sort(x) & \text{if } w = 0x \\ sort(x) \bullet 1 & \text{if } w = 1x \end{cases}$$

   (a) Prove that $\#(0, sort(w)) = \#(0, w)$ for every string $w \in \{0,1\}^*$.

   (b) Prove that $sort(w \bullet 1) = sort(w) \bullet 1$ for every string $w \in \{0,1\}^*$.

   (c) Prove that $sort(sort(w)) = sort(w)$ for every string $w \in \{0,1\}^*$.

   Think about these two problems on your own; do not submit solutions:

   (d) Prove that $x \bullet 0 \neq y \bullet 1$ for all strings $x, y \in \{0,1\}^*$.

   (e) Prove that $sort(w) \neq x \bullet 10 \bullet y$, for all strings $w, x, y \in \{0,1\}^*$.

   You may assume without proof that $\#(a, uv) = \#(a, u) + \#(a, v)$ for any symbol a and any strings u and v, or any other result proved in class, in lab, or in the lecture notes. Your proofs for later parts of this problem can assume earlier parts even if you don't prove them. Otherwise, your proofs must be formal and self-contained.

3. Consider the set of strings $L \subseteq \{0, 1\}^*$ defined recursively as follows:

   - The empty string ε is in L.
   - For any strings x in L, the strings $0x1$ and $1x0$ are also in L.
   - For any two *nonempty* strings x and y in L, the string $x \bullet y$ is also in L.
   - These are the only strings in L.

   This problem asks you to prove that L is the set of all strings w where the number of 0s is equal to the number of 1s. More formally, for any string w, let $\Delta(w) = \#(1, w) - \#(0, w)$, or equivalently,

$$\Delta(w) = \begin{cases} 0 & \text{if } w = \varepsilon \\ \Delta(x) - 1 & \text{if } w = 0x \\ \Delta(x) + 1 & \text{if } w = 1x \end{cases}$$

   (a) Prove that the string 11011100101000 is in L.

   (b) Prove that $\Delta(w) = 0$ for every string $w \in L$.

   (c) Prove that L contains every string $w \in \{0, 1\}^*$ such that $\Delta(w) = 0$.

   You can assume the following properties of the Δ function, for all strings w and z.

   - Addition: $\Delta(wz) = \Delta(w) + \Delta(z)$.
   - Downward interpolation: If $\Delta(wz) > 0$ and $\Delta(z) < 0$, then there are strings x and y such that $w = xy$ and $\Delta(yz) = 0$.
   - Upward interpolation: If $\Delta(wz) < 0$ and $\Delta(z) > 0$, then there are strings x and y such that $w = xy$ and $\Delta(yz) = 0$.

   The interpolation properties are a type of "intermediate value theorem". **Think about how to prove these properties yourself.**

   You can also assume any other result proved in class, in lab, or in the lecture notes. Otherwise, your proofs must be formal and self-contained.

Solved Problems

Each homework assignment will include at least one solved problem, similar to the problems assigned in that homework, together with the grading rubric we would apply if this problem appeared on a homework or exam. These model solutions illustrate our recommendations for structure, presentation, and level of detail in your homework solutions. Of course, the actual **content** *of your solutions won't match the model solutions, because your problems are different!*

4. The ***reversal*** $\boldsymbol{w^R}$ of a string w is defined recursively as follows:

$$w^R := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ x^R \bullet a & \text{if } w = a \cdot x \end{cases}$$

A ***palindrome*** is any string that is equal to its reversal, like AMANAPLANACANALPANAMA, RACECAR, POOP, I, and the empty string.

(a) Give a recursive definition of a palindrome over the alphabet Σ.

(b) Prove $w = w^R$ for every palindrome w (according to your recursive definition).

(c) Prove that every string w such that $w = w^R$ is a palindrome (according to your recursive definition).

You may assume without proof the following statements for all strings x, y, and z:

- Reversal reversal: $(x^R)^R = x$
- Concatenation reversal: $(x \bullet y)^R = y^R \bullet x^R$
- Right cancellation: If $x \bullet z = y \bullet z$, then $x = y$.

Solution:

(a) A string $w \in \Sigma^*$ is a palindrome if and only if either

- $w = \varepsilon$, or
- $w = a$ for some symbol $a \in \Sigma$, or
- $w = axa$ for some symbol $a \in \Sigma$ and some *palindrome* $x \in \Sigma^*$.

Rubric: 2 points = ½ for each base case + 1 for the recursive case. No credit for the rest of the problem unless this part is correct.

(b) Let w be an arbitrary palindrome.

Assume that $x = x^R$ for every palindrome x such that $|x| < |w|$.

There are three cases to consider (mirroring the definition of "palindrome"):

- If $w = \varepsilon$, then $w^R = \varepsilon$ by definition, so $w = w^R$.
- If $w = a$ for some symbol $a \in \Sigma$, then $w^R = a$ by definition, so $w = w^R$.
- Finally, if $w = axa$ for some symbol $a \in \Sigma$ and some palindrome $x \in P$,

then

$$
\begin{aligned}
w^R &= (a \cdot x \bullet a)^R \\
&= (x \bullet a)^R \bullet a && \text{by definition of reversal} \\
&= a^R \bullet x^R \bullet a && \text{by concatenation reversal} \\
&= a \bullet x^R \bullet a && \text{by definition of reversal} \\
&= a \bullet x \bullet a && \text{by the inductive hypothesis} \\
&= w && \text{by assumption}
\end{aligned}
$$

In all three cases, we conclude that $w = w^R$. ■

Rubric: 4 points: standard induction rubric (scaled)

(c) Let w be an arbitrary string such that $w = w^R$.
Assume that every string x such that $|x| < |w|$ and $x = x^R$ is a palindrome.
There are three cases to consider (mirroring the definition of "palindrome"):

- If $w = \varepsilon$, then w is a palindrome by definition.
- If $w = a$ for some symbol $a \in \Sigma$, then w is a palindrome by definition.
- Otherwise, we have $w = ax$ for some symbol a and some *non-empty* string x.
  The definition of reversal implies that $w^R = (ax)^R = x^R a$.
  Because x is non-empty, its reversal x^R is also non-empty.
  Thus, $x^R = by$ for some symbol b and some string y.
  It follows that $w^R = bya$, and therefore $w = (w^R)^R = (bya)^R = ay^R b$.

  [At this point, we need to prove that $a = b$ and that y is a palindrome.]

  Our assumption that $w = w^R$ implies that $bya = ay^R b$.
  The recursive definition of string equality immediately implies $a = b$.

  Because $a = b$, we have $w = ay^R a$ and $w^R = aya$.
  The recursive definition of string equality implies $y^R a = ya$.
  Right cancellation implies that $y^R = y$.
  The inductive hypothesis now implies that y is a palindrome.

  We conclude that w is a palindrome by definition.

In all three cases, we conclude that w is a palindrome. ■

Rubric: 4 points: standard induction rubric (scaled).

Standard induction rubric. For problems worth 10 points:

- + 1 for explicitly considering an *arbitrary* object.
- + 2 for a valid ***strong*** induction hypothesis
  - **Deadly Sin!** Automatic zero for stating a weak induction hypothesis, unless the rest of the proof is *absolutely perfect*.
- + 2 for explicit exhaustive case analysis
  - No credit here if the case analysis omits an infinite number of objects. (For example: all odd-length palindromes.)
  - −1 if the case analysis omits an finite number of objects. (For example: the empty string.)
  - −1 for making the reader infer the case conditions. Spell them out!
  - No penalty if the cases overlap (for example: even length at least 2, odd length at least 3, and length at most 5.)
- + 1 for cases that do not invoke the inductive hypothesis ("base cases")
  - No credit here if one or more "base cases" are missing.
- + 2 for correctly applying the ***stated*** inductive hypothesis
  - No credit here for applying a ***different*** inductive hypothesis, even if that different inductive hypothesis would be valid.
- + 2 for other details in cases that invoke the inductive hypothesis ("inductive cases")
  - No credit here if one or more "inductive cases" are missing.

For (sub)problems worth less than 10 points, scale and round to the nearest half-integer.

CS/ECE 374 A ✧ Fall 2019

Homework 1

Due Tuesday, September 10, 2019 at 8pm

Starting with this homework, groups of up to three people can submit joint solutions. Each problem should be submitted by exactly one person, and the beginning of the homework should clearly state the Gradescope names and email addresses of each group member. In addition, whoever submits the homework must tell Gradescope who their other group members are.

---

1. For each of the following languages over the alphabet $\{0, 1\}$, give a regular expression that describes that language, and *briefly* argue why your regular expression is correct.

   (a) All strings except `010`.

   (b) All strings that contain the substring `010`.

   (c) All strings that contain the subsequence `010`.

   (d) All strings that do not contain the substring `010`.

   (e) All strings that do not contain the subsequence `010`.

   (The technical terms "substring" and "subsequence" are defined in the lecture notes.)

2. Let L be the set of all strings in $\{0, 1\}^*$ that contain *at least two* occurrences of the substring `010`.

   (a) Give a regular expression for L, and briefly argue why your expression is correct.

   (b) Describe a DFA over the alphabet $\Sigma = \{0, 1\}$ that accepts the language L.

   You may either draw the DFA or describe it formally, but the states Q, the start state s, the accepting states A, and the transition function δ must be clearly specified. (See the standard DFA rubric for more details.)

   Argue that your DFA is correct by explaining what each state in your DFA *means*. Drawings or formal descriptions without English explanations will receive no credit, even if they are correct.

   [Hint: The shortest string in L has length 5.]

3. Let L denote the set of all strings $w \in \{0,1\}^*$ that satisfy *at most two* of the following conditions:

   - The substring 01 appears in w an odd number of times.
   - $\#(1,w)$ is divisible by 3.
   - The binary value of w is *not* a multiple of 7.

   For example: The string 00100101 satisfies all three conditions, so 00100011 is ***not*** in L, and the empty string ε satisfies only the second condition, so $\varepsilon \in L$. (01 appears in ε zero times, and the binary value of ε is 0, because what else could it be?)

   Formally describe a DFA with input alphabet $\Sigma = \{0,1\}$ that accepts the language L, by explicitly describing the states Q, the start state s, the accepting states A, and the transition function δ. Do not attempt to *draw* your DFA; the smallest DFA for this language has 84 states, which is *way* too many for a drawing to be understandable.

   Argue that your machine is correct by explaining what each state in your DFA *means*. Formal descriptions without English explanations will receive no credit, even if they are correct. (See the standard DFA rubric for more details.)

   This is an exercise in clear communication. We are not only asking you to design a *correct* DFA. We are also asking you to clearly, precisely, and convincingly explain your DFA to another human being who understands DFAs but has *not* thought about this particular problem. Excessive formality and excessive brevity will hurt you just as much as imprecision and handwaving.

Solved problem

4. ***C comments*** are the set of strings over alphabet $\Sigma = \{$`*`, `/`, `A`, `◇`, `↲`$\}$ that form a proper comment in the C program language and its descendants, like C++ and Java. Here `↲` represents the newline character, `◇` represents any other whitespace character (like the space and tab characters), and `A` represents any non-whitespace character other than `*` or `/`.[1] There are two types of C comments:

   - Line comments: Strings of the form `//`···`↲`
   - Block comments: Strings of the form `/*`···`*/`

   Following the C99 standard, we explicitly disallow ***nesting*** comments of the same type. A line comment starts with `//` and ends at the first `↲` after the opening `//`. A block comment starts with `/*` and ends at the the first `*/` completely after the opening `/*`; in particular, every block comment has at least two `*`s. For example, each of the following strings is a valid C comment:

   `/***/`   `//◇//◇↲`   `/*///◇*◇↲**/`   `/*◇//◇↲◇*/`

   On the other hand, *none* of the following strings is a valid C comment:

   `/*/`   `//◇//◇↲◇↲`   `/*◇/*◇*/◇*/`

   (Questions about C comments start on the next page.)

[1] The actual C commenting syntax is considerably more complex than described here, because of character and string literals.

- The opening `/*` or `//` of a comment must not be inside a string literal (`"`···`"`) or a (multi-)character literal (`'`···`'`).
- The opening double-quote of a string literal must not be inside a character literal (`'"'`) or a comment.
- The closing double-quote of a string literal must not be escaped (`\"`)
- The opening single-quote of a character literal must not be inside a string literal (`"`···`'`···`"`) or a comment.
- The closing single-quote of a character literal must not be escaped (`\'`)
- A backslash escapes the next symbol if and only if it is not itself escaped (`\\`) or inside a comment.

For example, the string `"/*\\\"*/"/*"/*\"/*"*/` is a valid string literal (representing the 5-character string `/*\"\*/`, which is itself a valid block comment!) followed immediately by a valid block comment. ***For this homework question, just pretend that the characters `'`, `"`, and `\` don't exist.***

Commenting in C++ is even more complicated, thanks to the addition of *raw* string literals. Don't ask.

Some C and C++ compilers do support nested block comments, in violation of the language specification. A few other languages, like OCaml, explicitly allow nesting block comments.

(a) Describe a regular expression for the set of all C comments.

Solution:

$$\texttt{//}(\texttt{/}+\texttt{*}+\texttt{A}+\diamond)^*\text{↲} \quad + \quad \texttt{/*}\Big(\texttt{/}+\texttt{A}+\diamond+\text{↲}+\texttt{*}\texttt{*}^*(\texttt{A}+\diamond+\text{↲})\Big)^* \;\texttt{*}^*\texttt{*/}$$

The first subexpression matches all line comments, and the second subexpression matches all block comments. Within a block comment, we can freely use any symbol other than $\texttt{*}$, but any run of $\texttt{*}$s must be followed by a character in $(\texttt{A}+\diamond+\text{↲})$ or by the closing slash of the comment. ■

Rubric: Standard regular expression rubric. This is not the only correct solution.

(b) Describe a regular expression for the set of all strings composed entirely of blanks ($\diamond$), newlines (↲), and C comments.

Solution:

$$\Big(\diamond+\text{↲} + \texttt{//}(\texttt{/}+\texttt{*}+\texttt{A}+\diamond)^*\text{↲}+\texttt{/*}(\texttt{/}+\texttt{A}+\diamond+\text{↲}+\texttt{*}\texttt{*}^*(\texttt{A}+\diamond+\text{↲}))^*\,\texttt{*}\texttt{*}^*\texttt{/}\Big)^*$$

This regular expression has the form $(\langle\text{whitespace}\rangle + \langle\text{comment}\rangle)^*$, where $\langle\text{whitespace}\rangle$ is the regular expression $\diamond+\text{↲}$ and $\langle\text{comment}\rangle$ is the regular expression from part (a). ■

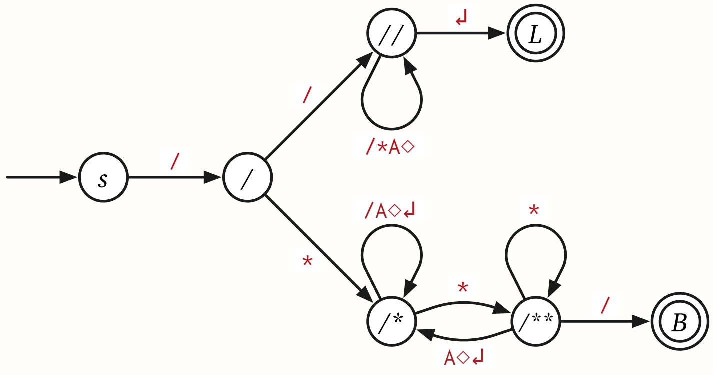
Rubric: Standard regular expression rubric. This is not the only correct solution.

Standard regular expression rubric. For problems worth 10 points:

- 2 points for a syntactically correct regular expression.
- **Homework only:** 4 points for a *brief* English explanation of your regular expression. This is how you argue that your regular expression is correct.
  - **Deadly Sin ("Declare your variables."): No credit for the problem if the English explanation is missing, *even if the regular expression is correct*.**
  - For longer expressions, you should explain each of the major components of your expression, and separately explain how those components fit together.
  - We do not want a *transcription*; don't just translate the regular-expression *notation* into English.
- 4 points for correctness. (8 points on exams, with all penalties doubled)
  - —1 for a single mistake: one typo, excluding exactly one string in the target language, or including exactly one string not in the target language.
  - —2 for incorrectly including/excluding more than one but a finite number of strings.
  - —4 for incorrectly including/excluding an infinite number of strings.
- Regular expressions that are more complex than necessary may be penalized. Regular expressions that are *significantly* too complex may get no credit at all. On the other hand, minimal regular expressions are *not* required for full credit.

(c) Describe a DFA that accepts the set of all C comments.

Solution: The following eight-state DFA recognizes the language of C comments. All missing transitions lead to a hidden reject state.



The states are labeled mnemonically as follows:

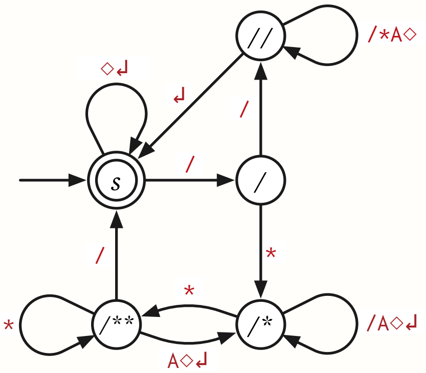
- *s* — We have not read anything.
- / — We just read the initial /.
- // — We are reading a line comment.
- *L* — We have just read a complete line comment.
- /* — We are reading a block comment, and we did not just read a * after the opening /*.
- /** — We are reading a block comment, and we just read a * after the opening /*.
- *B* — We have just read a complete block comment.

■

Rubric: Standard DFA design rubric. This is not the only correct solution, or even the simplest correct solution. (We don't need two distinct accepting states.)

(d) Describe a DFA that accepts the set of all strings composed entirely of blanks (◇), newlines (↲), and C comments.

Solution: By merging the accepting states of the previous DFA with the start state and adding white-space transitions at the start state, we obtain the following six-state DFA. Again, all missing transitions lead to a hidden reject state.



The states are labeled mnemonically as follows:

- *s* — We are between comments.
- / — We just read the initial / of a comment.
- // — We are reading a line comment.
- /* — We are reading a block comment, and we did not just read a * after the opening /*.
- /** — We are reading a block comment, and we just read a * after the opening /*.

■

Rubric: Standard DFA design rubric. This is not the only correct solution, but it is the simplest correct solution.

Standard DFA design rubric. For problems worth 10 points:

- 2 points for an unambiguous description of a DFA, including the states set Q, the start state s, the accepting states A, and the transition function δ.
    - **Drawings:** Use an arrow from nowhere to indicate s, and doubled circles to indicate accepting states A. If $A = \varnothing$, say so explicitly. If your drawing omits a junk/trash/reject state, say so explicitly. **Draw neatly!** If we can't read your solution, we can't give you credit for it.
    - **Text descriptions:** You can describe the transition function either using a 2d array, using mathematical notation, or using an algorithm.
    - **Product constructions:** You must give a complete description of each the DFAs you are combining (as either drawings, text, or recursive products), together with the accepting states of the product DFA.
- **Homework only:** 4 points for *briefly* explaining the purpose of each state *in English*. This is how you argue that your DFA is correct.
    - **Deadly Sin ("Declare your variables."): No credit for the problem if the English description is missing, *even if the DFA is correct*.**
    - For product constructions, explaining the states in the factor DFAs is both necessary and sufficient.
- 4 points for correctness. (8 points on exams, with all penalties doubled)
    - −1 for a single mistake: a single misdirected transition, a single missing or extra accepting state, rejecting exactly one string that should be accepted, or accepting exactly one string that should be accepted.
    - −2 for incorrectly accepting/rejecting more than one but a finite number of strings.
    - −4 for incorrectly accepting/rejecting an infinite number of strings.
- DFAs that are more complex than necessary may be penalized. DFAs that are *significantly* more complex than necessary may get no credit at all. On the other hand, *minimal* DFAs are *not* required for full credit, unless the problem explicitly asks for them.
- Half credit for describing an NFA when the problem asks for a DFA.

*5. Recall that the reversal w^R of a string w is defined recursively as follows:

$$w^R := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ x^R \bullet a & \text{if } w = a \cdot x \end{cases}$$

The reversal L^R of any *language* L is the set of reversals of all strings in L:

$$L^R := \left\{ w^R \;\middle|\; w \in L \right\}.$$

Prove that the reversal of every regular language is regular.

Solution: Let r be an arbitrary regular expression. We want to derive a regular expression r' such that $L(r') = L(r)^R$.

Assume for any proper subexpression s of s that there is a regular expression s' such that $L(s') = L(s)^R$.

There are five cases to consider (mirroring the definition of regular expressions).

(a) If $r = \varnothing$, then we set $r' = \varnothing$, so that

$$\begin{aligned} L(r)^R &= L(\varnothing)^R && \text{because } r = \varnothing \\ &= \varnothing^R && \text{because } L(\varnothing) = \varnothing \\ &= \varnothing && \text{because } \varnothing^R = \varnothing \\ &= L(\varnothing) && \text{because } L(\varnothing) = \varnothing \\ &= L(r') && \text{because } r = \varnothing \end{aligned}$$

(b) If $r = w$ for some string $w \in \Sigma^*$, then we set $r' := w^R$, so that

$$\begin{aligned} L(r)^R &= L(w)^R && \text{because } r = w \\ &= \{w\}^R && \text{because } L(\langle\mathsf{string}\rangle) = \{\langle\mathsf{string}\rangle\} \\ &= \{w^R\} && \text{by definition of } L^R \\ &= L(w^R) && \text{because } L(\langle\mathsf{string}\rangle) = \{\langle\mathsf{string}\rangle\} \\ &= L(r') && \text{because } r = w^R \end{aligned}$$

(c) Suppose $r = s^*$ for some regular expression s. The inductive hypothesis implies a regular expressions s' such that $L(s') = L(s)^R$. Let $\boldsymbol{r' = (s')^*}$; then we have

$$\begin{aligned} L(r)^R &= L(s^*)^R && \text{because } r = s^* \\ &= (L(s)^*)^R && \text{by definition of } {}^* \\ &= (L(s)^R)^* && \text{because } (L^R)^* = (L^*)^R \\ &= (L(s'))^* && \text{by definition of } s' \\ &= L((s')^*) && \text{by definition of } {}^* \\ &= L(r') && \text{by definition of } r' \end{aligned}$$

(d) Suppose $r = s + t$ for some regular expressions s and t. The inductive hypothesis implies regular expressions s' and t' such that $L(s') = L(s)^R$ and $L(t') = L(t)^R$.

Set $\boldsymbol{r' := s' + t'}$; then we have

$$\begin{aligned}
L(r)^R &= L(s+t)^R && \text{because } r = s+t\\
&= (L(s) \cup L(t))^R && \text{by definition of } +\\
&= \{w^R \mid w \in (L(s) \cup L(t))\} && \text{by definition of } L^R\\
&= \{w^R \mid w \in L(s) \text{ or } w \cup L(t)\} && \text{by definition of } \cup\\
&= \{w^R \mid w \in L(s)\} \;\cup\; \{w^R \mid w \cup L(t)\} && \text{by definition of } \cup\\
&= L(s)^R \cup L(t)^R && \text{by definition of } L^R\\
&= L(s') \cup L(t') && \text{by definition of } s' \text{ and } t'\\
&= L(s' + t') && \text{by definition of } +\\
&= L(r') && \text{by definition of } r'
\end{aligned}$$

(e) Suppose $r = s \bullet t$ for some regular expressions s and t. The inductive hypothesis implies regular expressions s' and t' such that $L(s') = L(s)^R$ and $L(t') = L(t)^R$. Set $\boldsymbol{r' = t's'}$; then we have

$$\begin{aligned}
L(r)^R &= L(st)^R && \text{because } r = s+t\\
&= (L(s) \bullet L(t))^R && \text{by definition of } \bullet\\
&= \{w^R \mid w \in (L(s) \bullet L(t))\} && \text{by definition of } L^R\\
&= \{(x \bullet y)^R \mid x \in L(s) \text{ and } y \in L(t)\} && \text{by definition of } \bullet\\
&= \{y^R \bullet x^R \mid x \in L(s) \text{ and } y \in L(t)\} && \text{concatenation reversal}\\
&= \{y' \bullet x' \mid x' \in L(s)^R \text{ and } y' \in L(t)^R\} && \text{by definition of } L^R\\
&= \{y' \bullet x' \mid x' \in L(s') \text{ and } y' \in L(t')\} && \text{by definition of } s' \text{ and } t'\\
&= L(t') \bullet L(s') && \text{by definition of } \bullet\\
&= L(t' \bullet s') && \text{by definition of } \bullet\\
&= L(r') && \text{by definition of } r'
\end{aligned}$$

In all five cases, we have found a regular expression r' such that $L(r') = L(r)^R$. It follows that $L(r)^R$ is regular. ■

Rubric: Standard induction rubric!!

CS/ECE 374 A ✧ Fall 2019

Homework 2

Due Tuesday, September 17, 2019 at 8pm

---

1. Prove that the following languages are *not* regular.

   (a) $\{0^m 1^n \mid m > n\}$

   (b) $\{w \in (0+1)^* \mid \#(0,w)/\#(1,w) \text{ is an integer}\}$ *[Hint: $n/0$ is never an integer.]*

   (c) The set of all palindromes in $(0+1)^*$ whose length is divisible by 7.

2. For each of the following regular expressions, describe or draw two finite-state machines:

   - An NFA that accepts the same language, constructed from the given regular expression using Thompson's algorithm (described in class and in the notes).
   - An equivalent DFA, constructed from your NFA using the incremental subset algorithm (described in class and in the notes). For each state in your DFA, identify the corresponding subset of states in your NFA. Your DFA should have no unreachable states.

   (a) $(0+11)^*(00+1)^*$

   (b) $(((0^*+1)^*+0)^*+1)^*$

3. For each of the following languages over the alphabet $\Sigma = \{0,1\}$, either prove that the language is regular (by constructing an appropriate DFA, NFA, or regular expression) or prove that the language is not regular (by constructing an infinite fooling set). Recall that Σ^+ denotes the set of all *nonempty* strings over Σ. Watch those parentheses!

   (a) $\{0^a 1^b 0^c \mid (a \le b+c \text{ and } b \le a+c) \text{ or } c \le a+b\}$

   (b) $\{0^a 1^b 0^c \mid a \le b+c \text{ and } (b \le a+c \text{ or } c \le a+b)\}$

   (c) $\{wxw^R \mid w,x \in \Sigma^+\}$

   (d) $\{ww^Rx \mid w,x \in \Sigma^+\}$

   [Hint: Exactly two of these languages are regular.]

Solved problem

4. For each of the following languages, either prove that the language is regular (by constructing an appropriate DFA, NFA, or regular expression) or prove that the language is not regular (by constructing an infinite fooling set).

   Recall that a *palindrome* is a string that equals its own reversal: $w = w^R$. Every string of length 0 or 1 is a palindrome.

   (a) Strings in $(0+1)^*$ in which no prefix of length at least 2 is a palindrome.

   Solution: Regular: $\varepsilon + 01^* + 10^*$. Call this language L_a.

   Let w be an arbitrary non-empty string in $(0+1)^*$. Without loss of generality, assume $w = 0x$ for some string x. There are two cases to consider.

   - If x contains a 0, then we can write $w = 01^n0y$ for some integer n and some string y. The prefix 01^n0 is a palindrome of length at least 2. Thus, $w \notin L_a$.
   - Otherwise, $x \in 1^*$. Every non-empty prefix of w is equal to 01^n for some non-negative integer $n \le |x|$. Every palindrome that starts with 0 also ends with 0, so the only palindrome prefixes of w are ε and 0, both of which have length less than 2. Thus, $w \in L_a$.

   We conclude that $0x \in L_a$ if and only if $x \in 1^*$. A similar argument implies that $1x \in L_a$ if and only if $x \in 0^*$. Finally, trivially, $\varepsilon \in L_a$. ■

   Rubric: 2½ points = ½ for "regular" + 1 for regular expression + 1 for justification. This is more detail than necessary for full credit.

   (b) Strings in $(0+1+2)^*$ in which no prefix of length at least 2 is a palindrome.

   Solution: Not regular. Call this language L_b.

   I claim that the infinite language $F = (012)^+$ is a fooling set for L_b.

   Let x and y be arbitrary distinct strings in F.

   Then $x = (012)^i$ and $y = (012)^j$ for some positive integers $i \neq j$.

   Without loss of generality, assume $i < j$.

   Let z be the suffix $(210)^i$.

   - $xz = (012)^i(210)^i$ is a palindrome of length $6i \ge 2$, so $xz \notin L_b$.
   - $yz = (012)^j(210)^i$ has no palindrome prefixes except ε and 0, because $i < j$, so $yz \in L_b$.

   We conclude that F is a fooling set for L_b, as claimed.

   Because F is infinite, L_b cannot be regular. ■

   Rubric: 2½ points = ½ for "not regular" + 2 for fooling set proof (standard rubric, scaled).

(c) Strings in $(0+1)^*$ in which no prefix of length at least 3 is a palindrome.

Solution: Not regular. Call this language L_c.

I claim that the infinite language $F = (001101)^+$ is a fooling set for L_c.

Let x and y be arbitrary distinct strings in F.

Then $x = (001101)^i$ and $y = (001101)^j$ for some positive integers $i \neq j$.

Without loss of generality, assume $i < j$.

Let z be the suffix $(101100)^i$.

- $xz = (001101)^i(101100)^i$ is a palindrome of length $12i \geq 2$, so $xz \notin L_b$.
- $yz = (001101)^j(101100)^i$ has no palindrome prefixes except ε and 0, because $i < j$, so $yz \in L_b$.

We conclude that F is a fooling set for L_c, as claimed.

Because F is infinite, L_c cannot be regular. ■

Rubric: 2½ points = ½ for "not regular" + 2 for fooling set proof (standard rubric, scaled).

(d) Strings in $(0+1)^*$ in which no *substring* of length at least 3 is a palindrome.

Solution: Regular. Call this language L_d.

Every palindrome of length at least 3 contains a palindrome substring of length 3 or 4. Thus, the complement language $\overline{L_d}$ is described by the regular expression

$$(0+1)^*(000+010+101+111+0110+1001)(0+1)^*$$

Thus, $\overline{L_d}$ is regular, so its complement L_d is also regular. ■

Solution: Regular. Call this language L_d.

In fact, L_d is *finite*! Appending either 0 or 1 to any of the underlined strings creates a palindrome suffix of length 3 or 4.

$$\varepsilon+0+1+00+01+10+11+001+\underline{011}+\underline{100}+110+\underline{0011}+\underline{1100}$$

■

Rubric: 2½ points = ½ for "regular" + 2 for proof:

- 1 for expression for $\overline{L_d}$ + 1 for applying closure
- 1 for regular expression + 1 for justification

Standard fooling set rubric. For problems worth 5 points:

- 2 points for the fooling set:
  - + 1 for explicitly describing the proposed fooling set F.
  - + 1 if the proposed set F is actually a fooling set for the target language.
  - — No credit for the proof if the proposed set is not a fooling set.
  - — No credit for the *problem* if the proposed set is finite.
- 3 points for the proof:
  - ◦ The proof must correctly consider *arbitrary* strings $x, y \in F$.
    - — No credit for the proof unless both x and y are *always* in F.
    - — No credit for the proof unless x and y can be *any* strings in F.
  - + 1 for correctly describing a suffix z that distinguishes x and y.
  - + 1 for proving either $xz \in L$ or $yz \in L$.
  - + 1 for proving either $yz \notin L$ or $xz \notin L$, respectively.

As usual, scale partial credit (rounded to nearest ½) for problems worth fewer points.

CS/ECE 374 A ✧ Fall 2019

Homework 3

Due Tuesday, September 24, 2018 at 8pm

This is the last homework before Midterm 1.

---

1. Describe context-free grammars for the following languages over the alphabet $\Sigma = \{\mathtt{0}, \mathtt{1}\}$. For each non-terminal in your grammars, describe in English the language generated by that non-terminal.

   (a) $\{\mathtt{0}^m \mathtt{1}^n \mid m > n\}$

   (b) The set of all palindromes in $(\mathtt{0} + \mathtt{1})^*$ whose length is divisible by 7.

   (c) $\left\{\mathtt{0}^a \mathtt{1}^b \;\middle|\; a \neq 2b \text{ and } b \neq 2a\right\}$

   [Hint: You proved that the first two languages are non-regular in HW2.1(a) and HW2.1(c). The language described in HW2.1(b) is not even context-free!]

2. For any string w, let $\mathit{contract}(w)$ denote the string obtained by collapsing each maximal substring of equal symbols to one symbol. For example:

$$\begin{aligned} \mathit{contract}(\mathtt{010101}) &= \mathtt{010101} \\ \mathit{contract}(\mathtt{001110}) &= \mathtt{010} \\ \mathit{contract}(\mathtt{111111}) &= \mathtt{1} \\ \mathit{contract}(\mathtt{1}) &= \mathtt{1} \\ \mathit{contract}(\varepsilon) &= \varepsilon \end{aligned}$$

   Prove that for every regular language L over the alphabet $\{\mathtt{0}, \mathtt{1}\}$, the following languages are also regular:

   (a) $\mathit{contract}(L) = \{\mathit{contract}(w) \mid w \in L\}$

   (b) $\mathit{contract}^{-1}(L) = \{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid \mathit{contract}(w) \in L\}$

3. For any string w, let $\mathit{oneswap}(w)$ be the set of all strings obtained by swapping exactly one pair of adjacent symbols in w. For example:

$$\begin{aligned} \mathit{oneswap}(\mathtt{010101}) &= \{\mathtt{100101}, \mathtt{001101}, \mathtt{011001}, \mathtt{010011}, \mathtt{010110}\} \\ \mathit{oneswap}(\mathtt{001110}) &= \{\mathtt{001110}, \mathtt{010110}, \mathtt{001101}\} \\ \mathit{oneswap}(\mathtt{111111}) &= \{\mathtt{111111}\} \\ \mathit{oneswap}(\mathtt{1}) &= \varnothing \\ \mathit{oneswap}(\varepsilon) &= \varnothing \end{aligned}$$

   For any language L, define a new language $\mathit{oneswap}(L)$ as follows:

$$\mathit{oneswap}(L) := \bigcup_{w \in L} \mathit{oneswap}(w)$$

   Prove that if L is a regular language over the alphabet $\{\mathtt{0}, \mathtt{1}\}$, the language $\mathit{oneswap}(L)$ is also regular.

Solved problem

4. (a) Fix an arbitrary regular language L. Prove that the language $\mathit{half}(L) := \{w \mid ww \in L\}$ is also regular.

Solution: Let $M = (\Sigma, Q, s, A, \delta)$ be an arbitrary DFA that accepts L. We define a new NFA $M' = (\Sigma, Q', s', A', \delta')$ with ε-transitions that accepts $\mathit{half}(L)$, as follows:

$$\begin{aligned} Q' &= (Q \times Q \times Q) \cup \{s'\} \\ s' &\text{ is an explicit state in } Q' \\ A' &= \{(h,h,q) \mid h \in Q \text{ and } q \in A\} \end{aligned}$$

$$\begin{aligned} \delta'(s', \varepsilon) &= \{(s,h,h) \mid h \in Q\} \\ \delta'(s', a) &= \varnothing \end{aligned}$$

$$\begin{aligned} \delta'((p,h,q), \varepsilon) &= \varnothing \\ \delta'((p,h,q), a) &= \big\{\big(\delta(p,a), h, \delta(q,a)\big)\big\} \end{aligned}$$

M' reads its input string w and simulates M reading the input string ww. Specifically, M' simultaneously simulates two copies of M, one reading the left half of ww starting at the usual start state s, and the other reading the right half of ww starting at some intermediate state h.

- The new start state s' non-deterministically guesses the "halfway" state $h = \delta^*(s,w)$ without reading any input; this is the only non-determinism in M'.
- State (p,h,q) means the following:
  - The left copy of M (which started at state s) is now in state p.
  - The initial guess for the halfway state is h.
  - The right copy of M (which started at state h) is now in state q.
- M' accepts if and only if the left copy of M ends at state h (so the initial non-deterministic guess $h = \delta^*(s,w)$ was correct) and the right copy of M ends in an accepting state.

■

Solution (smartass): A complete solution is given in the lecture notes. ■

Rubric: 5 points: standard langage transformation rubric (scaled). Yes, the smartass solution would be worth full credit.

(b) Describe a regular language L such that the language $double(L) := \{ww \mid w \in L\}$ is *not* regular. Prove your answer is correct.

Solution: Consider the regular language $L = 0^*1$.

Expanding the regular expression lets us rewrite $L = \{0^n1 \mid n \ge 0\}$. It follows that $double(L) = \{0^n10^n1 \mid n \ge 0\}$. I claim that this language is not regular.

Let x and y be arbitrary distinct strings in L.

Then $x = 0^i1$ and $y = 0^j1$ for some integers $i \ne j$.

Then x is a distinguishing suffix of these two strings, because

- $xx \in double(L)$ by definition, but
- $yx = 0^i10^j1 \notin double(L)$ because $i \ne j$.

We conclude that L is a fooling set for $double(L)$.

Because L is infinite, $double(L)$ cannot be regular. ■

Solution: Consider the regular language $L = \Sigma^* = (0 + 1)^*$.

I claim that the language $double(\Sigma^*) = \{ww \mid w \in \Sigma^*\}$ is not regular.

Let F be the infinite language 01^*0.

Let x and y be arbitrary distinct strings in F.

Then $x = 01^i0$ and $y = 01^j0$ for some integers $i \ne j$.

The string $z = 1^i$ is a distinguishing suffix of these two strings, because

- $xz = 01^i01^i = ww$ where $w = 01^i$, so $xz \in double(\Sigma^*)$, but
- $yx = 01^j01^i \notin double(\Sigma^*)$ because $i \ne j$.

We conclude that F is a fooling set for $double(\Sigma^*)$.

Because F is infinite, $double(\Sigma^*)$ cannot be regular. ■

Rubric: 5 points:

- 2 points for describing a regular language L such that $double(L)$ is not regular.
- 1 point for describing an infinite fooling set for $double(L)$:
  - + ½ for explicitly describing the proposed fooling set F.
  - + ½ if the proposed set F is actually a fooling set.
  - — No credit for the proof if the proposed set is not a fooling set.
  - — No credit for the *problem* if the proposed set is finite.
- 2 points for the proof:
  - + ½ for correctly considering *arbitrary* strings x and y
    - — No credit for the proof unless both x and y are *always* in F.
    - — No credit for the proof unless both x and y can be *any* string in F.
  - + ½ for correctly stating a suffix z that distinguishes x and y.
  - + ½ for proving either $xz \in L$ or $yz \in L$.
  - + ½ for proving either $yz \notin L$ or $xz \notin L$, respectively.

These are not the only correct solutions. These are not the only fooling sets for these languages.

Standard langage transformation rubric. For problems worth 10 points:

- + 2 for a formal, complete, and unambiguous description of the output automaton, including the states, the start state, the accepting states, and the transition function, as functions of an *arbitrary* input DFA. The description must state whether the output automaton is a DFA, an NFA without ε-transitions, or an NFA with ε-transitions.
  - No points for the rest of the problem if this is missing.
- + 2 for a *brief* English explanation of the output automaton. We explicitly do *not* want a formal proof of correctness, or an English *transcription*, but a few sentences explaining how your machine works and justifying its correctness. What is the overall idea? What do the states represent? What is the transition function doing? Why these accepting states?
  - **Deadly Sin:** No points for the rest of the problem if this is missing.
- + 6 for correctness
  - + 3 for accepting *all* strings in the target language
  - + 3 for accepting *only* strings in the target language
  - – 1 for a single mistake in the formal description (for example a typo)
  - Double-check correctness when the input language is $\varnothing$, or $\{\varepsilon\}$, or $\mathtt{0}^*$, or Σ^*.

CS/ECE 374 A ✧ Fall 2019

Homework 4

Due Tuesday, October 8, 2019 at 8pm

1. The following variant of the infamous StoogeSort algorithm[1] was discovered by the British actor Patrick Troughton during rehearsals for the 20th anniversary *Doctor Who* special "The Five Doctors".[2]

```
WhoSort($A[1\,..\,n]$):
  if $n < 13$
      sort $A$ by brute force
  else
      $k = \lceil n/5 \rceil$
      WhoSort($A[1\,..\,3k]$)          ⟨⟨Hartnell⟩⟩
      WhoSort($A[2k+1\,..\,n]$)        ⟨⟨Troughton⟩⟩
      WhoSort($A[1\,..\,3k]$)          ⟨⟨Pertwee⟩⟩
      WhoSort($A[k+1\,..\,4k]$)        ⟨⟨Davison⟩⟩
```

   (a) Prove by induction that WhoSort correctly sorts its input. *[Hint: Where can the smallest k elements be?]*

   (b) Would WhoSort still sort correctly if we replaced "if $n < 13$" with "if $n < 4$"? Justify your answer.

   (c) Would WhoSort still sort correctly if we replaced "$k = \lceil n/5 \rceil$" with "$k = \lfloor n/5 \rfloor$"? Justify your answer.

   (d) What is the running time of WhoSort? (Set up a running-time recurrence and then solve it, ignoring the floors and ceilings.)

2. In the lab on Wednesday, we developed an algorithm to compute the median of the union of two sorted arrays size n in $O(\log n)$ time.

   But now suppose we are given *three* sorted arrays $A[1\,..\,n]$, $B[1\,..\,n]$, and $C[1\,..\,n]$. Describe and analyze an algorithm to compute the median of $A \cup B \cup C$ in $O(\log n)$ time. (You can assume the arrays contain $3n$ distinct integers.)

[1] https://en.wikipedia.org/wiki/Stooge_sort

[2] Tom Baker, the fourth Doctor, declined to return for the reunion; hence, only four Doctors appeared in "The Five Doctors". (Well, okay, technically the BBC used excerpts of the unfinished episode "Shada" to include Baker, but he wasn't really *there*—to the extent that any fictional character in a television show about a time traveling wizard arguing with several other versions of himself about immortality can be said to be "really" "there".)

3. At the end of the second act of the action blockbuster *Fast and Impossible XIII¾: Guardians of Expendable Justice Reloaded*, the villainous Dr. Metaphor hypnotizes the entire Hero League/Force/Squad, arranges them in a long line at the edge of a cliff, and instructs each hero to shoot the closest taller heroes to their left and right, at a prearranged signal.

   Suppose we are given the heights of all n heroes, in order from left to right, in an array $Ht[1\,..\,n]$. (To avoid salary arguments, the producers insisted that no two heroes have the same height.) Then we can compute the Left and Right targets of each hero in $O(n^2)$ time using the following algorithm.

```
WhoTargetsWhom($Ht[1\,..\,n]$):
  for $j \gets 1$ to $n$
      ⟨⟨Find the left target $L[j]$ for hero $j$⟩⟩
      $L[j] \gets$ None
      for $i \gets 1$ to $j-1$
          if $Ht[i] > Ht[j]$
              $L[j] \gets i$
      ⟨⟨Find the right target $R[j]$ for hero $j$⟩⟩
      $R[j] \gets$ None
      for $k \gets n$ down to $j+1$
          if $Ht[k] > Ht[j]$
              $R[j] \gets k$
  return $L[1\,..\,n]$, $R[1\,..\,n]$
```

   (a) Describe a divide-and-conquer algorithm that computes the output of WhoTargetsWhom in $O(n \log n)$ time.

   (b) Prove that at least $\lfloor n/2 \rfloor$ of the n heroes are targets. That is, prove that the output arrays $R[0\,..\,n-1]$ and $L[0\,..\,n-1]$ contain at least $\lfloor n/2 \rfloor$ distinct values (other than None).

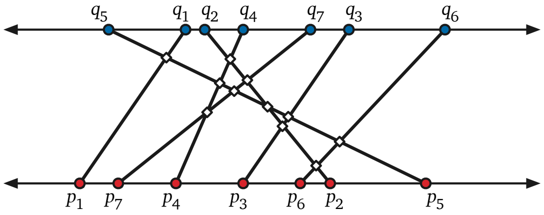
   (c) Alas, Dr. Metaphor's diabolical plan is successful. At the prearranged signal, all the heroes simultaneously shoot their targets, and all targets fall over the cliff, apparently dead. Metaphor repeats his dastardly experiment over and over; after each massacre, he forces the remaining heroes to choose new targets, following the same algorithm, and then shoot their targets at the next signal. Eventually, only the shortest member of the Hero Crew/Alliance/Posse is left alive.[3]

       Describe an algorithm that computes the number of rounds before Dr. Metaphor's deadly process finally ends. For full credit, your algorithm should run in $\boldsymbol{O(n)}$ time.

[3] In the thrilling final act, Retcon the Squirrel, the last surviving member of the Hero Team/Group/Society (played by Tom Baker, of course), saves everyone by traveling back in time and retroactively replacing the other $n-1$ heroes with lifelike balloon sculptures. So, yeah, it's basically *Avengers: Endgame* meets *Doom Patrol*.

Solved problem

4. Suppose we are given two sets of n points, one set $\{p_1, p_2, \ldots, p_n\}$ on the line $y = 0$ and the other set $\{q_1, q_2, \ldots, q_n\}$ on the line $y = 1$. Consider the n line segments connecting each point p_i to the corresponding point q_i. Describe and analyze a divide-and-conquer algorithm to determine how many pairs of these line segments intersect, in $O(n \log n)$ time. See the example below.



Seven segments with endpoints on parallel lines, with 11 intersecting pairs.

Your input consists of two arrays $P[1..n]$ and $Q[1..n]$ of x-coordinates; you may assume that all $2n$ of these numbers are distinct. No proof of correctness is necessary, but you should justify the running time.

Solution: We begin by sorting the array $P[1..n]$ and permuting the array $Q[1..n]$ to maintain correspondence between endpoints, in $O(n \log n)$ time. Then for any indices $i < j$, segments i and j intersect if and only if $Q[i] > Q[j]$. Thus, our goal is to compute the number of pairs of indices $i < j$ such that $Q[i] > Q[j]$. Such a pair is called an ***inversion***.

We count the number of inversions in Q using the following extension of mergesort; as a side effect, this algorithm also sorts Q. If $n < 100$, we use brute force in $O(1)$ time. Otherwise:

- Color the elements in the Left half $Q[1..\lfloor n/2 \rfloor]$ bLue.
- Color the elements in the Right half $Q[\lfloor n/2 \rfloor + 1..n]$ Red.
- Recursively count inversions in (and sort) the blue subarray $Q[1..\lfloor n/2 \rfloor]$.
- Recursively count inversions in (and sort) the red subarray $Q[\lfloor n/2 \rfloor + 1..n]$.
- Count red/blue inversions as follows:
  - MERGE the sorted subarrays $Q[1..n/2]$ and $Q[n/2 + 1..n]$, maintaining the element colors.
  - For each blue element $Q[i]$ of the now-sorted array $Q[1..n]$, count the number of smaller red elements $Q[j]$.

The last substep can be performed in $O(n)$ time using a simple for-loop:

```
CountRedBlue($A[1\,..\,n]$):
    $count \gets 0$
    $total \gets 0$
    for $i \gets 1$ to $n$
        if $A[i]$ is red
            $count \gets count + 1$
        else
            $total \gets total + count$
    return $total$
```

Merge and CountRedBlue each run in $O(n)$ time. Thus, the running time of our inversion-counting algorithm obeys the mergesort recurrence $T(n) = 2T(n/2) + O(n)$. (We can safely ignore the floors and ceilings in the recursive arguments.) We conclude that the overall running time of our algorithm is $O(n \log n)$, as required.

Rubric: This is enough for full credit.

In fact, we can execute the third merge-and-count step directly by modifying the Merge algorithm, without any need for "colors". Here changes to the standard Merge algorithm are indicated in red.

```
MergeAndCount($A[1\,..\,n], m$):
    $i \gets 1$;  $j \gets m + 1$;  $count \gets 0$;  $total \gets 0$
    for $k \gets 1$ to $n$
        if $j > n$
            $B[k] \gets A[i]$;  $i \gets i + 1$;  $total \gets total + count$
        else if $i > m$
            $B[k] \gets A[j]$;  $j \gets j + 1$;  $count \gets count + 1$
        else if $A[i] < A[j]$
            $B[k] \gets A[i]$;  $i \gets i + 1$;  $total \gets total + count$
        else
            $B[k] \gets A[j]$;  $j \gets j + 1$;  $count \gets count + 1$
    for $k \gets 1$ to $n$
        $A[k] \gets B[k]$
    return $total$
```

We can further optimize MergeAndCount by observing that *count* is always equal to $j - m - 1$, so we don't need an additional variable. (Proof: Initially, $j = m + 1$ and $count = 0$, and we always increment j and *count* together.)

```
MergeAndCount2($A[1\,..\,n], m$):
  $i \gets 1$;  $j \gets m+1$;  $total \gets 0$
  for $k \gets 1$ to $n$
      if $j > n$
          $B[k] \gets A[i]$;  $i \gets i+1$;  $total \gets total + \boldsymbol{j - m - 1}$
      else if $i > m$
          $B[k] \gets A[j]$;  $j \gets j+1$
      else if $A[i] < A[j]$
          $B[k] \gets A[i]$;  $i \gets i+1$;  $total \gets total + \boldsymbol{j - m - 1}$
      else
          $B[k] \gets A[j]$;  $j \gets j+1$
  for $k \gets 1$ to $n$
      $A[k] \gets B[k]$
  return $total$
```

MergeAndCount2 still runs in $O(n)$ time, so the overall running time is still $O(n \log n)$, as required. ■

Rubric: 10 points = 2 for base case + 3 for divide (split and recurse) + 3 for conquer (merge and count) + 2 for time analysis. Max 3 points for a correct $O(n^2)$-time algorithm. This is neither the only way to correctly describe this algorithm nor the only correct $O(n \log n)$-time algorithm. No proof of correctness is required.

Notice that each boxed algorithm is preceded by an English description of the task that algorithm performs. **Omitting these descriptions is a Deadly Sin.**

CS/ECE 374 A ✧ Fall 2019

Homework 5

Due Tuesday, October 15, 2019 at 8pm

---

1. Farmers Boggis, Bunce, and Bean have set up an obstacle course for Mr. Fox. The course consists of a long row of booths, each with a number painted on the front with bright red paint. Formally, Mr. Fox is given an array $A[1\,..\,n]$, where $A[i]$ is the number painted on the front of the ith booth. Each number $A[i]$ could be positive, negative, or zero. Everyone agrees with the following rules:

   - Mr. Fox must visit all the booths in order from 1 to n.
   - At each booth, Mr. Fox must say one word: either "Ring!" or "Ding!"
   - If Mr. Fox says "Ring!" at the ith booth, he earns a reward of $A[i]$ chickens. (If $A[i] < 0$, Mr. Fox pays a penalty of $-A[i]$ chickens.)
   - If Mr. Fox says "Ding!" at the ith booth, he pays a penalty of $A[i]$ chickens. (If $A[i] < 0$, Mr. Fox earns a reward of $-A[i]$ chickens.)
   - Mr. Fox is forbidden to say the same word more than three times in a row. For example, if he says "Ring!" at booths 6, 7, and 8, then he must say "Ding!" at booth 9.
   - All accounts will be settled at the end, after Mr. Fox visits every booth and the umpire calls "Hot box!" Mr. Fox does not actually have to carry chickens (or anti-chickens) through the obstacle course.
   - Finally, if Mr. Fox violates any of the rules, or if he ends the obstacle course owing the farmers chickens, the farmers will shoot him.

   Describe and analyze an algorithm to compute the largest number of chickens that Mr. Fox can earn by running the obstacle course, given the array $A[1\,..\,n]$ of numbers as input. *[Hint: Watch out for the burning pine cone!]*

2. Recall that a *supersequence* of a string w is any string obtained from w by inserting zero or more symbols. For example, the strings STRING, STIRRING, and MISTERFINNIGAN are all supersequences of the string STRING.

   (a) Recall that a *palindrome* is a string that is equal to its reversal, like the empty string, A, HANNAH, or AMANAPLANACATACANALPANAMA. Describe an algorithm to compute the length of the shortest palindrome supersequence of a given string.

   (b) A *dromedrome* is an even-length string whose first half is equal to its second half, like the empty string, AA, ACKACK, or AMANAPLANAMANAPLAN. Describe an algorithm to compute the length of the shortest dromedrome supersequence of a given string.

   For example, given the string SUPERSEQUENCE as input, your algorithm for part (a) should return 21 (the length of SUPECNRSEQUQESRNCEPUS), and your algorithm for part (b) should return 20 (the length of SEQUPERNCESEQUPERNCE). The input to both algorithms is an array $A[1\,..\,n]$ representing a string.

3. Suppose you are given a DFA M with k states for Jeff's favorite regular language $L \subseteq (\mathtt{0}+\mathtt{1})^*$.

   (a) Describe and analyze an algorithm that decides whether a given bit-string belongs to the language L^*.

   (b) Describe and analyze an algorithm that partitions a given bit-string into as many substrings as possible, such that L contains every substring in the partition. Your algorithm should return only the number of substrings, not their actual positions. (In light of your algorithm from part (a), you can assume that an appropriate partition exists.)

   For example, suppose L is the set of all bit-strings that start and end with `1` and whose length is *not* divisible by 3.[1] Then given the input string `10111100001101010110111101`, your algorithm for part (a) should return True, and your algorithm for part (b) should return the integer 5, which is the length of the following partition:

   $$\mathtt{1011} \bullet \mathtt{1} \bullet \mathtt{10000110101} \bullet \mathtt{1011} \bullet \mathtt{1101}$$

   The input to both algorithms consists of (some reasonable representation of) the DFA M and an array $A[1..n]$ of bits. Express the running time of your algorithms as functions of both k (the number of states in M) and n (the length of the input string).

   *[Hint: Do **not** try to build a DFA for L^*.]*

[1] This is not actually Jeff's favorite regular language.

Solved Problem

4. A *shuffle* of two strings X and Y is formed by interspersing the characters into a new string, keeping the characters of X and Y in the same order. For example, the string BANANAANANAS is a shuffle of the strings BANANA and ANANAS in several different ways.

   BANANAANANAS BANANAANANAS BANANAANANAS

   Similarly, the strings PRODGYRNAMAMMIINCG and DYPRONGARMAMMICING are both shuffles of DYNAMIC and PROGRAMMING:

   PRODGYRNAMAMMIINCG DYPRONGARMAMMICING

   Given three strings $A[1..m]$, $B[1..n]$, and $C[1..m+n]$, describe and analyze an algorithm to determine whether C is a shuffle of A and B.

Solution: We define a boolean function $\mathit{Shuf}(i, j)$, which is TRUE if and only if the prefix $C[1..i+j]$ is a shuffle of the prefixes $A[1..i]$ and $B[1..j]$. This function satisfies the following recurrence:

$$\mathit{Shuf}(i,j) = \begin{cases} \textsc{True} & \text{if } i = j = 0 \\ \mathit{Shuf}(0, j-1) \land (B[j] = C[j]) & \text{if } i = 0 \text{ and } j > 0 \\ \mathit{Shuf}(i-1, 0) \land (A[i] = C[i]) & \text{if } i > 0 \text{ and } j = 0 \\ \big(\mathit{Shuf}(i-1, j) \land (A[i] = C[i+j])\big) & \\ \qquad \lor \big(\mathit{Shuf}(i, j-1) \land (B[j] = C[i+j])\big) & \text{if } i > 0 \text{ and } j > 0 \end{cases}$$

We need to compute $\mathit{Shuf}(m, n)$.

We can memoize all function values into a two-dimensional array $\mathit{Shuf}[0..m][0..n]$. Each array entry $\mathit{Shuf}[i, j]$ depends only on the entries immediately below and immediately to the right: $\mathit{Shuf}[i-1, j]$ and $\mathit{Shuf}[i, j-1]$. Thus, we can fill the array in standard row-major order. The original recurrence gives us the following pseudocode:

```
SHUFFLE?($A[1..m]$, $B[1..n]$, $C[1..m+n]$):
  $Shuf[0,0] \gets$ TRUE
  for $j \gets 1$ to $n$
      $Shuf[0,j] \gets Shuf[0,j-1] \land (B[j] = C[j])$
  for $i \gets 1$ to $n$
      $Shuf[i,0] \gets Shuf[i-1,0] \land (A[i] = B[i])$
      for $j \gets 1$ to $n$
          $Shuf[i,j] \gets$ FALSE
          if $A[i] = C[i+j]$
              $Shuf[i,j] \gets Shuf[i,j] \lor Shuf[i-1,j]$
          if $B[i] = C[i+j]$
              $Shuf[i,j] \gets Shuf[i,j] \lor Shuf[i,j-1]$
  return $Shuf[m,n]$
```

The algorithm runs in ***O*(*mn*) *time***. ■

Rubric: Max 10 points: Standard dynamic programming rubric. No proofs required. Max 7 points for a slower polynomial-time algorithm; scale partial credit accordingly.

Standard dynamic programming rubric. For problems worth 10 points:

- 6 points for a correct recurrence, described either using mathematical notation or as pseudocode for a recursive algorithm.
  - + 1 point for a clear English description of the function you are trying to evaluate. (Otherwise, we don't even know what you're *trying* to do.) **Deadly Sin: Automatic zero if the English description is missing.**
  - + 1 point for stating how to call your function to get the final answer.
  - + 1 point for base case(s). −½ for one *minor* bug, like a typo or an off-by-one error.
  - + 3 points for recursive case(s). −1 for each *minor* bug, like a typo or an off-by-one error. **No credit for the rest of the problem if the recursive case(s) are incorrect.**
- 4 points for details of the dynamic programming algorithm
  - + 1 point for describing the memoization data structure
  - + 2 points for describing a correct evaluation order; a clear picture is usually sufficient. If you use nested loops, be sure to specify the nesting order.
  - + 1 point for time analysis
- It is *not* necessary to state a space bound.
- For problems that ask for an algorithm that computes an optimal *structure*—such as a subset, partition, subsequence, or tree—an algorithm that computes only the *value* or *cost* of the optimal structure is sufficient for full credit, unless the problem specifically says otherwise.
- Official solutions usually include pseudocode for the final iterative dynamic programming algorithm, ***but iterative pseudocode is not required for full credit***. If your solution includes iterative pseudocode, you do not need to separately describe the recurrence, memoization structure, or evaluation order. But you ***do*** still need to describe the underlying recursive function in English.
- Official solutions will provide target time bounds. Algorithms that are faster than this target are worth more points; slower algorithms are worth fewer points, typically by 2 or 3 points (out of 10) for each factor of n. Partial credit is scaled to the new maximum score, and all points above 10 are recorded as extra credit.

  We rarely include these target time bounds in the actual questions, because when we have included them, significantly more students turned in algorithms that meet the target time bound but didn't work (earning 0/10) instead of correct algorithms that are slower than the target time bound (earning 8/10).

CS/ECE 374 A ✧ Fall 2019

Homework 6

Due Tuesday, October 22, 2019 at 8pm

---

1. A non-empty sequence $S[1\,..\,\ell]$ of positive integers is called a ***perfect ruler sequence*** if it satisfies the following conditions:

   - The length of S is one less than a power of 2; that is, $\ell = 2^k - 1$ for some integer k.
   - Let $m = \lceil \ell/2 \rceil = 2^{k-1}$. Then $S[m]$ is the unique maximum element of S.
   - If $\ell > 1$, then the prefix $S[1\,..\,m-1]$ is a perfect ruler sequence.
   - If $\ell > 1$, then the suffix $S[m+1\,..\,\ell]$ is a perfect ruler sequence.

   For example, the following sequence is a perfect ruler sequence:

   $$\langle 2, 7, 6, 9, 5, 8, 5, 12, 1, 9, 4, 10, 7, 8, 3 \rangle$$

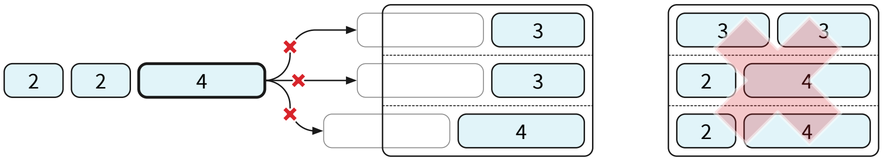
   Describe and analyze an efficient algorithm to compute the longest perfect ruler subsequence of a given array $A[1\,..\,n]$ of integers.

2. Suppose you are running a ferry across Lake Michigan.[1] The vehicle hold in your ferry is L meters long and three lanes wide. As each vehicle drives up to your ferry, you direct it to one of the three lanes; the vehicle then parks as far forward in that lane as possible. Vehicles must enter the ferry in the order they arrived; if the vehicle at the front of the queue doesn't fit into any of the lanes, then no more vehicles are allowed to board.

   Because your uncle runs the concession stand at the ferry terminal, you want to load as *few* vehicles onto your ferry as possible for each trip. But you don't want to be *obvious* about it, if the vehicle at the front of the queue fits anywhere, you must assign it to a lane where it fits. You can see the lengths of all vehicles in the queue on your security camera.

   Describe and analyze an algorithm to load the ferry. The input to your algorithm is the integer L and an array $len[1\,..\,n]$ containing the (integer) lengths of all vehicles in the queue. (You can assume that $1 \le len[i] \le L$ for all i.) Your output should be the *smallest* integer k such that you can put vehicles 1 through k onto the ferry, in such a way that vehicle $k+1$ does not fit. Express the running time of your algorithm as a function of both n (the number of vehicles) and L (the length of the ferry).

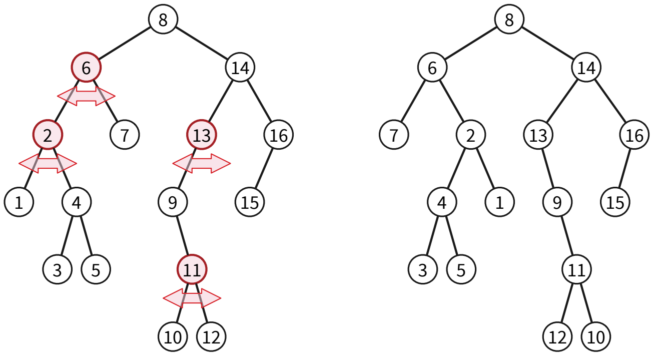
   For example, suppose $L = 6$, and the first six vehicles in the queue have lengths 3, 3, 4, 4, 2, and 2. Your algorithm should return the integer 3, because if you assign the first three vehicles to three different lanes, the fourth vehicle won't fit. (A different lane assignment gets all six vehicles on board, but that would rob your uncle of three customers.)



[1] Welcome aboard the Recursion Ferry! How we get across the water is none of your business!

3. CS 125 students Chef Gallon and Fade Waygone wrote some inorder traversal code for their MP on binary search trees. To keep things simple, they wisely chose the integers 1 through n as their search keys. Unfortunately, their code contained a subtle bug (which was nearly impossible to track down, thanks to version inconsistencies between Fade's laptop, the submission/grading server, and Oracle's ridiculous licencing terms) that would sporadically swap left and right child pointers in some binary tree nodes. As a result, their traversal code rarely returned the search keys in sorted order.

   For example, given the binary search tree below, if the four marked nodes had their left and right pointers swapped, Chef and Fade's traversal code would return the garbled "inorder" sequence $7, 6, 3, 4, 5, 2, 1, 8, 13, 9, 12, 11, 10, 14, 15, 16$.



   Chef and Fade submitted the output of several garbled traversals, but before they could submit the actual traversal code, Fade's laptop was infested with bees. After receiving a grade of 0 on their MP, Chef and Fade argued with their instructor that they should get *some* partial credit, because the sequences their code produced were at least consistent with correct binary search trees, and anyway the bees weren't their fault.

   Design and analyze an efficient algorithm to verify or refute Chef and Fade's claim (about the binary search trees, not the bees). The input to your algorithm is an array $A[1..n]$ containing a permutation of the integers 1 through n. Your algorithm should output TRUE if this array is the inorder traversal of an actual binary search tree with keys 1 through n, possibly with some left and right child pointers swapped, and FALSE otherwise. For example, if the input array contains $[5, 2, 3, 4, 1]$, your algorithm should return TRUE, and if the input array contains $[2, 5, 3, 1, 4]$, your algorithm should return FALSE.

Solved Problems

4. A string w of parentheses (and) and brackets [and] is ***balanced*** if and only if w is generated by the following context-free grammar:

$$S \to \varepsilon \mid (S) \mid [S] \mid SS$$

For example, the string $w = ([()][]())[()()]()$ is balanced, because $w = xy$, where

$$x = ([()][]()) \qquad \text{and} \qquad y = [()()]().$$

Describe and analyze an algorithm to compute the length of a longest balanced subsequence of a given string of parentheses and brackets. Your input is an array $A[1..n]$, where $A[i] \in \{(,),[,]\}$ for every index i.

Solution: Suppose $A[1..n]$ is the input string. For all indices i and k, let $\boldsymbol{LBS(i,k)}$ denote the length of the longest balanced subsequence of the substring $A[i..k]$. We need to compute $LBS(1,n)$. This function obeys the following recurrence:

$$LBS(i,j) = \begin{cases} 0 & \text{if } i \geq k \\ \max\left\{\begin{array}{c} 2 + LBS(i+1,k-1) \\ \max\limits_{j=1}^{k-1}\big(LBS(i,j) + LBS(j+1,k)\big) \end{array}\right\} & \text{if } A[i] \sim A[k] \\ \max\limits_{j=1}^{k-1}\big(LBS(i,j) + LBS(j+1,k)\big) & \text{otherwise} \end{cases}$$

Here $\boldsymbol{A[i] \sim A[k]}$ indicates that $A[i]$ and $A[k]$ are matching delimiters: Either $A[i] =$ (and $A[k] =$) or $A[i] =$ [and $A[k] =$].

We can memoize this function into a two-dimensional array $LBS[1..n, 1..n]$. Since every entry $LBS[i,j]$ depends only on entries in later rows or earlier columns (or both), we can evaluate this array row-by-row from bottom up in the outer loop, scanning each row from left to right in the inner loop. The resulting algorithm runs in $\boldsymbol{O(n^3)}$ ***time***.

```
LongestBalancedSubsequence($A[1..n]$):
  for $i \gets n$ down to 1
      $LBS[i,i] \gets 0$
      for $k \gets i+1$ to $n$
          if $A[i] \sim A[k]$
              $LBS[i,k] \gets LBS[i+1,k-1] + 2$
          else
              $LBS[i,k] \gets 0$
          for $j \gets i$ to $k-1$
              $LBS[i,k] \gets \max\{LBS[i,k],\ LBS[i,j] + LBS[j+1,k]\}$
  return $LBS[1,n]$
```

■

Rubric: 10 points, standard dynamic programming rubric

5. Oh, no! You've just been appointed as the new organizer of Giggle, Inc.'s annual mandatory holiday party! The employees at Giggle are organized into a strict hierarchy, that is, a tree with the company president at the root. The all-knowing oracles in Human Resources have assigned a real number to each employee measuring how "fun" the employee is. In order to keep things social, there is one restriction on the guest list: An employee cannot attend the party if their immediate supervisor is also present. On the other hand, the president of the company *must* attend the party, even though she has a negative fun rating; it's her company, after all.

   Describe an algorithm that makes a guest list for the party that maximizes the sum of the "fun" ratings of the guests. The input to your algorithm is a rooted tree T describing the company hierarchy, where each node v has a field $v.fun$ storing the "fun" rating of the corresponding employee.

Solution (two functions): We define two functions over the nodes of T.

- $\mathit{MaxFunYes}(v)$ is the maximum total "fun" of a legal party among the descendants of v, where v is definitely invited.
- $\mathit{MaxFunNo}(v)$ is the maximum total "fun" of a legal party among the descendants of v, where v is definitely not invited.

We need to compute $\mathit{MaxFunYes}(root)$. These two functions obey the following mutual recurrences:

$$\mathit{MaxFunYes}(v) = v.fun + \sum_{\text{children } w \text{ of } v} \mathit{MaxFunNo}(w)$$
$$\mathit{MaxFunNo}(v) = \sum_{\text{children } w \text{ of } v} \max\{\mathit{MaxFunYes}(w), \mathit{MaxFunNo}(w)\}$$

(These recurrences do not require separate base cases, because $\sum \varnothing = 0$.) We can memoize these functions by adding two additional fields $v.yes$ and $v.no$ to each node v in the tree. The values at each node depend only on the vales at its children, so we can compute all $2n$ values using a postorder traversal of T.

```
BestParty($T$):
    ComputeMaxFun($T.root$)
    return $T.root.yes$
```

```
ComputeMaxFun($v$):
    $v.yes \leftarrow v.fun$
    $v.no \leftarrow 0$
    for all children $w$ of $v$
        ComputeMaxFun($w$)
        $v.yes \leftarrow v.yes + w.no$
        $v.no \leftarrow v.no + \max\{w.yes, w.no\}$
```

(Yes, this is still dynamic programming; we're only traversing the tree recursively because that's the most natural way to traverse trees![a]) The algorithm spends $O(1)$ time at each node, and therefore runs in ***$O(n)$ time*** altogether. ■

[a] A naïve recursive implementation would run in $O(\phi^n)$ time in the worst case, where $\phi = (1+\sqrt{5})/2 \approx 1.618$ is the golden ratio. The worst-case tree is a path—every non-leaf node has exactly one child.

Solution (one function): For each node v in the input tree T, let $\mathit{MaxFun}(v)$ denote the maximum total "fun" of a legal party among the descendants of v, where v may or may not be invited.

The president of the company must be invited, so none of the president's "children" in T can be invited. Thus, the value we need to compute is

$$\mathit{root.fun} + \sum_{\text{grandchildren } w \text{ of } \mathit{root}} \mathit{MaxFun}(w).$$

The function MaxFun obeys the following recurrence:

$$\mathit{MaxFun}(v) = \max \left\{ \begin{array}{c} v.\mathit{fun} + \displaystyle\sum_{\text{grandchildren } x \text{ of } v} \mathit{MaxFun}(x) \\ \displaystyle\sum_{\text{children } w \text{ of } v} \mathit{MaxFun}(w) \end{array} \right\}$$

(This recurrence does not require a separate base case, because $\sum \varnothing = 0$.) We can memoize this function by adding an additional field $v.\mathit{maxFun}$ to each node v in the tree. The value at each node depends only on the values at its children and grandchildren, so we can compute all values using a postorder traversal of T.

```
BestParty(T):
  ComputeMaxFun(T.root)
  party ← T.root.fun
  for all children w of T.root
      for all children x of w
          party ← party + x.maxFun
  return party
```

```
ComputeMaxFun(v):
  yes ← v.fun
  no ← 0
  for all children w of v
      ComputeMaxFun(w)
      no ← no + w.maxFun
      for all children x of w
          yes ← yes + x.maxFun
  v.maxFun ← max{yes, no}
```

(Yes, this is still dynamic programming; we're only traversing the tree recursively because that's the most natural way to traverse trees![a])

The algorithm spends $O(1)$ time at each node (because each node has exactly one parent and one grandparent) and therefore runs in ***O(n) time*** altogether. ■

[a] Like the previous solution, a direct recursive implementation would run in $O(\phi^n)$ time in the worst case, where $\phi = (1+\sqrt{5})/2 \approx 1.618$ is the golden ratio.

Rubric: 10 points: standard dynamic programming rubric. These are not the only correct solutions.

CS/ECE 374 A ✧ Fall 2019

Homework 7

Due Tuesday, October 29, 2019 at 8pm

---

1. You new job at Object Oriented Parcel Service is to help direct delivery drivers through the city of Gridville. You are given a complete street map, in the form of a graph G, whose vertices are intersections, and whose edges represent streets between those intersections. Every street in Gridville runs in a straight line either north-south or east-west, and there are no one-way streets. One specific vertex s of G represents the OOPS warehouse.

   To increase fuel economy, decrease delivery times, and reduce accidents, OOPS imposes the following strict policies on its drivers.[1]

   - U-turns are forbidden, except at dead ends, where they are obviously required.
   - Left turns are forbidden, except where the road turns left, with no option to continue either straight or right.
   - Drivers must stop at every intersection.
   - Drivers must must park as close as possible to their destination address.

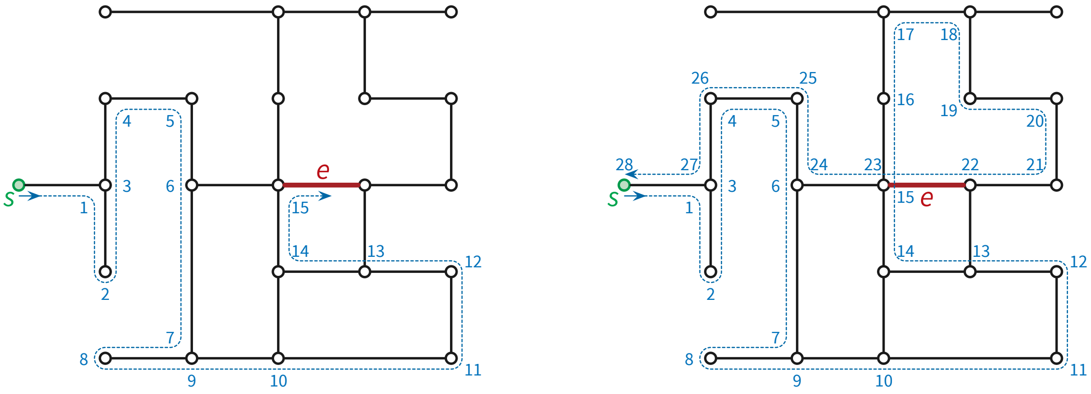
   Your job is to find routes from the OOPS warehouse to other locations in Gridville, with the smallest possible number of stops, that satisfy OOPS's driving policies. A destination is specified by an *edge* of G.

   (a) Describe and analyze an algorithm to find a legal route with the minimum number of stops from the OOPS warehouse to an arbitrary destination address. The input to your algorithm is the graph G, the start vertex s, and the destination edge; the output is the number of stops on the best legal route (or ∞ if there is no legal route).

   (b) After submitting your fancy new algorithm to your boss, you gently remind her that trucks have to return to the warehouse after making each delivery. Describe and analyze an algorithm to find a legal route with the minimum number of stops, from the OOPS warehouse, to an arbitrary destination address, and then back to the warehouse. The input to your algorithm is the graph G, the start vertex s, and the destination edge; the output is the number of stops on the best legal route (or ∞ if there is no legal route).

   For example, given the map on the next page, your algorithm for part (a) should return 15, and your algorithm for part (b) should return 28. Both optimal routes start with a forced right turn, followed by a forced U-turn, because turning left at a T intersection is forbidden. Notice that the optimal route to the destination is *not* a prefix of the optimal route to the destination and back.

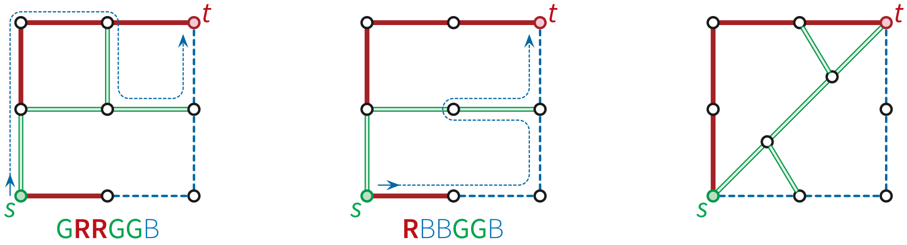
[1] OOPS maintains GPS trackers on every truck. If a driver ever breaks any of these rules, the tracker immediately shuts down and locks the truck, trapping the driver until a manager arrives, unlocks the truck, fires the driver, and installs a new replacement driver. OOPS managers are notoriously lazy, so most drivers keep enough food and water in their trucks to last several days.



Optimal delivery routes for OOPS drivers.

2. Suppose you are given an undirected graph G in which every edge is either red, green, or blue, along with two vertices s and t. Call a walk from s to t *awesome* if the walk does not contain three consecutive edges with the same color.

   Describe and analyze an algorithm to find the length of the shortest awesome walk from s to t. For example, given either the left or middle input below, your algorithm should return the integer 6, and given the input on the right, your algorithm should return ∞.



3. You are helping a group of ethnographers analyze some oral history data. The ethnographers have collected information about the lifespans of n different people, all now deceased, arbitrarily labeled with the integers 1 through n. Specifically, for some pairs (i, j), the ethnographers have learned one of the following facts:

   (a) Person i died before person j was born.

   (b) Person i and person j were both alive at some moment.

   The ethnographers are not sure that their facts are correct; after all, this information was passed down by word of mouth over generations, and memory is notoriously unreliable. So they would like you to determine whether the data they have collected is at least internally consistent, meaning there could have been people whose births and deaths consistent with their data.

   Describe and analyze an algorithm to answer the ethnographers' problem. Your algorithm should either output possible dates of birth and death that are consistent with all the stated facts, or it should report correctly that no such dates exist.

Solved Problem

4. Professor McClane takes you out to a lake and hands you three empty jars. Each jar holds a positive integer number of gallons; the capacities of the three jars may or may not be different. The professor then demands that you put exactly k gallons of water into one of the jars (which one doesn't matter), for some integer k, using only the following operations:

   (a) Fill a jar with water from the lake until the jar is full.

   (b) Empty a jar of water by pouring water into the lake.

   (c) Pour water from one jar to another, until either the first jar is empty or the second jar is full, whichever happens first.

   For example, suppose your jars hold 6, 10, and 15 gallons. Then you can put 13 gallons of water into the third jar in six steps:

   - Fill the third jar from the lake.
   - Fill the first jar from the third jar. (Now the third jar holds 9 gallons.)
   - Empty the first jar into the lake.
   - Fill the second jar from the lake.
   - Fill the first jar from the second jar. (Now the second jar holds 4 gallons.)
   - Empty the second jar into the third jar.

   Describe and analyze an efficient algorithm that either finds the smallest number of operations that leave exactly k gallons in any jar, or reports correctly that obtaining exactly k gallons of water is impossible. Your input consists of the capacities of the three jars and the positive integer k. For example, given the four numbers 6, 10, 15, and 13 as input, your algorithm should return the number 6 (the length of the sequence of operations listed above).

Solution: Let A, B, C denote the capacities of the three jars. We reduce the problem to breadth-first search in the following directed graph:

- $V = \big\{(a,b,c) \bigm| 0 \le a \le A \text{ and } 0 \le b \le B \text{ and } 0 \le c \le C\big\}$. Each vertex corresponds to a possible ***configuration*** of water in the three jars. There are $(A+1)(B+1)(C+1) = O(ABC)$ vertices altogether.
- The graph has a directed edge $(a,b,c) \to (a', b'c')$ whenever it is possible to move from the first configuration to the second in one step. Specifically, there is an edge from (a,b,c) to each of the following vertices (except those already equal to (a,b,c)):
  - $(0,b,c)$ and $(a,0,c)$ and $(a,b,0)$ — dumping a jar into the lake
  - (A,b,c) and (a,B,c) and (a,b,C) — filling a jar from the lake
  - $\left\{\begin{matrix} (0, a+b, c) & \text{if } a+b \le B \\ (a+b-B, B, c) & \text{if } a+b \ge B \end{matrix}\right\}$ — pouring from jar 1 into jar 2
  - $\left\{\begin{matrix} (0, b, a+c) & \text{if } a+c \le C \\ (a+c-C, b, C) & \text{if } a+c \ge C \end{matrix}\right\}$ — pouring from jar 1 into jar 3

- $\left\{\begin{matrix}(a+b,0,c) & \text{if } a+b \le A \\ (A,a+b-A,c) & \text{if } a+b \ge A\end{matrix}\right\}$ — pouring from jar 2 into jar 1
- $\left\{\begin{matrix}(a,0,b+c) & \text{if } b+c \le C \\ (a,b+c-C,C) & \text{if } b+c \ge C\end{matrix}\right\}$ — pouring from jar 2 into jar 3
- $\left\{\begin{matrix}(a+c,b,0) & \text{if } a+c \le A \\ (A,b,a+c-A) & \text{if } a+c \ge A\end{matrix}\right\}$ — pouring from jar 3 into Jar 1
- $\left\{\begin{matrix}(a,b+c,0) & \text{if } b+c \le B \\ (a,B,b+c-B) & \text{if } b+c \ge B\end{matrix}\right\}$ — pouring from jar 3 into jar 2

Since each vertex has at most 12 outgoing edges, there are at most $12(A+1)\times(B+1)(C+1) = O(ABC)$ edges altogether.

To solve the jars problem, we need to find the ***shortest path*** in G from the start vertex $(0,0,0)$ to any target vertex of the form $(k,\cdot,\cdot)$ or $(\cdot,k,\cdot)$ or $(\cdot,\cdot,k)$. We can compute this shortest path by calling ***breadth-first search*** starting at $(0,0,0)$, and then examining every target vertex by brute force. If BFS does not visit any target vertex, we report that no legal sequence of moves exists. Otherwise, we find the target vertex closest to $(0,0,0)$ and trace its parent pointers back to $(0,0,0)$ to determine the shortest sequence of moves. The resulting algorithm runs in $O(V+E) = \boldsymbol{O(ABC)}$ ***time***.

We can make this algorithm faster by observing that every move either leaves at least one jar empty or leaves at least one jar full. Thus, we only need vertices (a,b,c) where either $a=0$ or $b=0$ or $c=0$ or $a=A$ or $b=B$ or $c=C$; no other vertices are reachable from $(0,0,0)$. The number of non-redundant vertices and edges is $O(AB+BC+AC)$. Thus, if we only construct and search the relevant portion of G, the algorithm runs in $\boldsymbol{O(AB+BC+AC)}$ ***time***. ■

Rubric: 10 points: standard graph reduction rubric (see next page)

- Brute force construction is fine.

−1 for calling Dijkstra instead of BFS

- max 8 points for $O(ABC)$ time; scale partial credit.

Standard rubric for graph reduction problems. For problems out of 10 points:

+ 1 for correct vertices, *including English explanation for each vertex*
+ 1 for correct edges
  — ½ for forgetting "directed" if the graph is directed
+ 1 for stating the correct problem (in this case, "shortest path")
  — "Breadth-first search" is not a problem; it's an algorithm!
+ 1 for correctly applying the correct algorithm (in this case, "breadth-first search from $(0, 0, 0)$ and then examine every target vertex")
  — ½ for using a slower or more specific algorithm than necessary
+ 1 for time analysis in terms of the input parameters.
+ 5 for other details of the reduction
  - If your graph is constructed by naive brute force, you do not need to describe the construction algorithm; in this case, points for vertices, edges, problem, algorithm, and running time are all doubled.
  - Otherwise, apply the appropriate rubric, *including Deadly Sins*, to the construction algorithm. For example, for a solution that uses dynamic programming to build the graph quickly, apply the standard dynamic programming rubric.

CS/ECE 374 A ✧ Fall 2019

Homework 8

Due Tuesday, November 5, 2019 at 8pm

This is the last homework before Midterm 2.

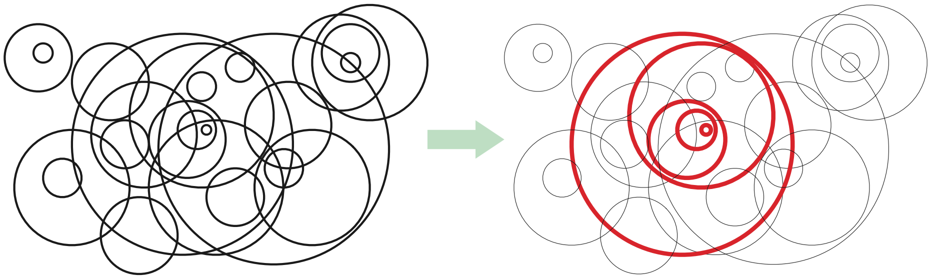
---

1. Over the summer, you pick up a part-time consulting gig at a gun range, designing targets for their customers to shoot at. A *target* consists of a properly nested set of circles, meaning for any two circles in the set, the smaller circle lies entirely inside the larger circle. The score for each shot is equal to the number of circles that contain the bullet hole.

   The shooters at the range aren't very good; their shots tend to be distributed uniformly at random on the target sheet. As a result, the expected score for any shot is proportional to the sum of the areas of the circles that make up the target. You'd like to make this expected score as large as possible.

   Now suppose your boss hands you a target sheet with n circles drawn on it. Describe and analyze an efficient algorithm to find a properly nested subset of these circles that maximizes the sum of the circle areas. You cannot *move* the circles; you must keep them exactly where your boss has drawn them.

   The input to your algorithm consists of three arrays $R[1\,..\,n]$, $X[1\,..\,n]$, and $Y[1\,..\,n]$, specifying the radius of each circle and the x- and y-coordinates of its center. The output is the sum of the circle areas in the best target.



2. The Cheery Hells neighborhood of Sham-Poobanana runs a popular and well-regulated Halloween celebration, attended by thousands of costumed children from all across Poobanana County. To regulate and protect the flood of costumed children, the Cheery Hells Neighborhood Association has designated a walking direction for each stretch of sidewalk.

   After paying the $25 entrance fee, each child receives a complete map of the neighborhood, in the form of a directed graph G, whose vertices represent houses. Each edge $v \rightarrow w$ indicates that one can walk directly from house v to house w following the designated sidewalk directions. (Anyone caught walking backward along a sidewalk is summarily ejected from Cheery Hells, without their candy. No refunds.) One special vertex s designates the entrance to Cheery Hells. Children can visit houses as many times as they like, but biometric scanners at every house ensure that each child receives candy only at their *first* visit to each house.

Unknown to the Neighborhood Association, the children of Cheery Hells have published a secret web site, accessible only through a link embedded in yet another TikTok cover of "Spooky Scary Skeletons", listing the amount of candy that each house in Cheery Hells will give to each visitor. (The web site also asks visitors to say "Gimme some Skittles, but I don't wanna pay for them" instead of "Trick or treat", just to mess with the grownups.)

Describe and analyze an algorithm to compute the maximum amount of candy that a single child can obtain in a walk through Cheery Hells, starting at the entrance node s. The input to your algorithm is the directed graph G, along with a non-negative integer $c(v)$ for each vertex describing the amount of candy that house gives each first-time visitor.

[Hint: Think about two special cases first: (1) Cheery Hells is strongly connected, and (2) Cheery Hells is acyclic. Solving only these two special cases is worth half credit.]

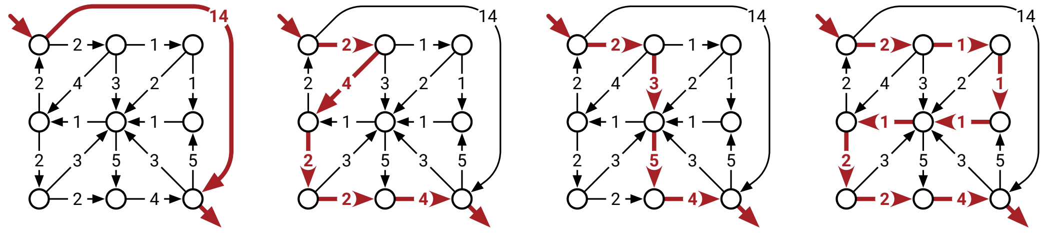
3. Morty needs to retrieve a stabilized plumbus from the Clackspire Labyrinth. He must enter the labyrinth using Rick's interdimensional portal gun, traverse the Labyrinth to a plumbus, then take that plumbus through the Labyrinth to a fleeb to be stabilized, and finally take the stabilized plumbus back to the original portal to return home. Plumbuses are stabilized by fleeb juice, which any fleeb will release immediately after being removed from its fleebhole. An unstabilized plumbus will explode if it is carried more than 137 flinks from its original storage unit. The Clackspire Labyrinth smells like farts, so Morty wants to spend as little time there as possible.

   Rick has given Morty a detailed map of the Clackspire Labyrinth, which consist of a directed graph $G = (V, E)$ with non-negative edge weights (indicating distance in flinks), along with two disjoint subsets $P \subset V$ and $F \subset V$, indicating the plumbus storage units and fleebholes, respectively. Morty needs to identify a start vertex s, a plumbus storage unit $p \in P$, and a fleebhole $f \in F$, such that the shortest-path distance from p to f is at most 137 flinks long, and the length of the shortest walk $s \rightsquigarrow p \rightsquigarrow f \rightsquigarrow s$ is as short as possible.

   Describe and analyze an algo(burp)rithm to so(burp)olve Morty's problem. You can assume that it is in fact possible for Morty to succeed.

Solved Problem

4. Although we typically speak of "the" shortest path from one vertex to another, a single graph could contain several minimum-length paths with the same endpoints.



Four (of many) equal-length shortest paths.

Describe and analyze an algorithm to compute the *number* of shortest paths from a source vertex s to a target vertex t in an arbitrary directed graph G with weighted edges. Assume that all edge weights are positive and that any necessary arithmetic operations can be performed in $O(1)$ time each.

[Hint: Compute shortest path distances from s to every other vertex. Throw away all edges that cannot be part of a shortest path from s to another vertex. What's left?]

Solution: We start by computing shortest-path distances $dist(v)$ from s to v, for every vertex v, using Dijkstra's algorithm. Call an edge $u{\to}v$ ***tight*** if $dist(u) + w(u{\to}v) = dist(v)$. Every edge in a shortest path from s to t must be tight. Conversely, every path from s to t that uses only tight edges has total length $dist(t)$ and is therefore a shortest path!

Let H be the subgraph of all tight edges in G. We can easily construct H in $O(V + E)$ time. Because all edge weights are positive, H is a directed acyclic graph. It remains only to count the number of paths from s to t in H.

For any vertex v, let $NumPaths(v)$ denote the number of paths in H from v to t; we need to compute $NumPaths(s)$. This function satisfies the following simple recurrence:

$$NumPaths(v) = \begin{cases} 1 & \text{if } v = t \\ \displaystyle\sum_{v\to w} NumPaths(w) & \text{otherwise} \end{cases}$$

In particular, if v is a sink but $v \neq t$ (and thus there are no paths from v to t), this recurrence correctly gives us $NumPaths(v) = \sum \varnothing = 0$.

We can memoize this function into the graph itself, storing each value $NumPaths(v)$ at the corresponding vertex v. Since each subproblem depends only on its successors in H, we can compute $NumPaths(v)$ for all vertices v by considering the vertices in reverse topological order, or equivalently, by performing a depth-first search of H starting at s. The resulting algorithm runs in $O(V + E)$ time.

The overall running time of the algorithm is dominated by Dijkstra's algorithm in the preprocessing phase, which runs in $\boldsymbol{O(E \log V)}$ ***time***. ■

Rubric: 10 points = 5 points for reduction to counting paths in a dag (standard graph reduction rubric) + 5 points for the path-counting algorithm (standard dynamic programming rubric)

CS/ECE 374 A ✧ Fall 2019

Homework 9

Due Tuesday, November 19, 2019 at 8pm

---

1. This problem asks you to develop polynomial-time algorithms for two (apparently) minor variants of 3Sat.

   (a) The input to **2Sat** is a boolean formula Φ in conjunctive normal form, with exactly **two** literals per clause, and the 2Sat problem asks whether there is an assignment to the variables of Φ such that every clause contains at least one True literal.

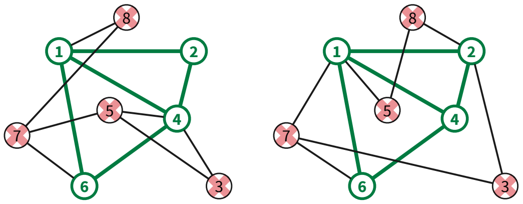
   Describe a polynomial-time algorithm for 2Sat. *[Hint: This problem is strongly connected to topics covered earlier in the semester.]*

   (b) The input to **Majority3Sat** is a boolean formula Φ in conjunctive normal form, with exactly three literals per clause. Majority3Sat asks whether there is an assignment to the variables of Φ such that every clause contains *at least two* True literals.

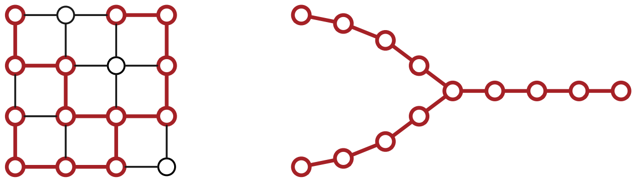
   Describe and analyze a polynomial-time reduction from Majority3Sat to 2Sat. Don't forget to prove that your reduction is correct.

   (c) Combining parts (a) and (b) gives us an algorithm for Majority3Sat. What is the running time of this algorithm?

2. Suppose we are given two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ with the same set of vertices $V = \{1, 2, \ldots, n\}$. Prove that is it NP-hard to find the smallest subset $S \subseteq V$ of vertices whose deletion leaves identical subgraphs $G_1 \setminus S = G_2 \setminus S$. For example, given the graphs below, the smallest subset has size 4.



3. A ***wye*** is an undirected graph that looks like the capital letter Y. More formally, a wye consists of three paths of equal length with one common endpoint, called the *hub*.



This grid graph contains a wye whose paths have length 4.

Prove that the following problem is NP-hard: Given an undirected graph G, what is the largest wye that is a subgraph of G? The three paths of the wye must not share any vertices except the hub, and they must have exactly the same length.

Solved Problem

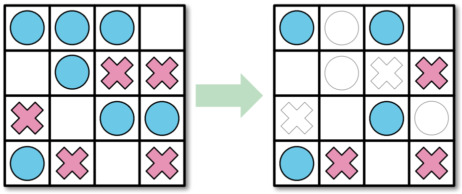
4. Consider the following solitaire game. The puzzle consists of an $n \times m$ grid of squares, where each square may be empty, occupied by a red stone, or occupied by a blue stone. The goal of the puzzle is to remove some of the given stones so that the remaining stones satisfy two conditions:

   (1) Every row contains at least one stone.

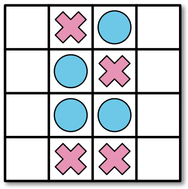
   (2) No column contains stones of both colors.

   For some initial configurations of stones, reaching this goal is impossible; see the example below.

   Prove that it is NP-hard to determine, given an initial configuration of red and blue stones, whether this puzzle can be solved.



A solvable puzzle and one of its many solutions.



An unsolvable puzzle.

Solution: We show that this puzzle is NP-hard by describing a reduction from 3SAT.

Let Φ be a 3CNF boolean formula with m variables and n clauses. We transform this formula into a puzzle configuration in polynomial time as follows. The size of the board is $n \times m$. The stones are placed as follows, for all indices i and j:

- If the variable x_j appears in the ith clause of Φ, we place a blue stone at (i, j).
- If the negated variable $\overline{x_j}$ appears in the ith clause of Φ, we place a red stone at (i, j).
- Otherwise, we leave cell (i, j) blank.

We claim that this puzzle has a solution if and only if Φ is satisfiable. This claim immediately implies that solving the puzzle is NP-hard. We prove our claim as follows:

$\Longrightarrow$ First, suppose Φ is satisfiable; consider an arbitrary satisfying assignment. For each index j, remove stones from column j according to the value assigned to x_j:

- If $x_j = \text{True}$, remove all red stones from column j.
- If $x_j = \text{False}$, remove all blue stones from column j.

In other words, remove precisely the stones that correspond to False literals. Because every variable appears in at least one clause, each column now contains stones of only one color (if any). On the other hand, each clause of Φ must contain at least one True literal, and thus each row still contains at least one stone. We conclude that the puzzle is satisfiable.

$\Longleftarrow$ On the other hand, suppose the puzzle is solvable; consider an arbitrary solution. For each index j, assign a value to x_j depending on the colors of stones left in column j:

- If column j contains blue stones, set $x_j = \text{True}$.
- If column j contains red stones, set $x_j = \text{False}$.
- If column j is empty, set x_j arbitrarily.

In other words, assign values to the variables so that the literals corresponding to the remaining stones are all True. Each row still has at least one stone, so each clause of Φ contains at least one True literal, so this assignment makes $\Phi = \text{True}$. We conclude that Φ is satisfiable.

This reduction clearly requires only polynomial time. ■

Rubric (Standard polynomial-time reduction rubric): 10 points =

+ 3 points for the reduction itself
  - For an NP-hardness proof, the reduction must be from a known NP-hard problem. You can use any of the NP-hard problems listed in the lecture notes (except the one you are trying to prove NP-hard, of course). **See the list on the next page.**
+ 3 points for the "if" proof of correctness
+ 3 points for the "only if" proof of correctness
+ 1 point for writing "polynomial time"

- An incorrect polynomial-time reduction that still satisfies half of the correctness proof is worth at most 4/10.
- A reduction in the wrong direction is worth 0/10.

Some useful NP-hard problems. You are welcome to use any of these in your own NP-hardness proofs, except of course for the specific problem you are trying to prove NP-hard.

CircuitSat: Given a boolean circuit, are there any input values that make the circuit output True?

3Sat: Given a boolean formula in conjunctive normal form, with exactly three distinct literals per clause, does the formula have a satisfying assignment?

MaxIndependentSet: Given an undirected graph G, what is the size of the largest subset of vertices in G that have no edges among them?

MaxClique: Given an undirected graph G, what is the size of the largest complete subgraph of G?

MinVertexCover: Given an undirected graph G, what is the size of the smallest subset of vertices that touch every edge in G?

MinSetCover: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subcollection whose union is S?

MinHittingSet: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subset of S that intersects every subset S_i?

3Color: Given an undirected graph G, can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

HamiltonianPath: Given graph G (either directed or undirected), is there a path in G that visits every vertex exactly once?

HamiltonianCycle: Given a graph G (either directed or undirected), is there a cycle in G that visits every vertex exactly once?

TravelingSalesman: Given a graph G (either directed or undirected) with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in G?

LongestPath: Given a graph G (either directed or undirected, possibly with weighted edges), what is the length of the longest simple path in G?

SteinerTree: Given an undirected graph G with some of the vertices marked, what is the minimum number of edges in a subtree of G that contains every marked vertex?

SubsetSum: Given a set X of positive integers and an integer k, does X have a subset whose elements sum to k?

Partition: Given a set X of positive integers, can X be partitioned into two subsets with the same sum?

3Partition: Given a set X of $3n$ positive integers, can X be partitioned into n three-element subsets, all with the same sum?

IntegerLinearProgramming: Given a matrix $A \in \mathbb{Z}^{n\times d}$ and two vectors $b \in \mathbb{Z}^n$ and $c \in Z^d$, compute $\max\{c \cdot x \mid Ax \leq b, x \geq 0, x \in \mathbb{Z}^d\}$.

FeasibleILP: Given a matrix $A \in \mathbb{Z}^{n\times d}$ and a vector $b \in \mathbb{Z}^n$, determine whether the set of feasible integer points $\max\{x \in \mathbb{Z}^d \mid Ax \leq b, x \geq 0\}$ is empty.

Draughts: Given an $n \times n$ international draughts configuration, what is the largest number of pieces that can (and therefore must) be captured in a single move?

SuperMarioBrothers: Given an $n \times n$ Super Mario Brothers level, can Mario reach the castle?

SteamedHams: Aurora borealis? At this time of year, at this time of day, in this part of the country, localized entirely within your kitchen? May I see it?

CS/ECE 374 A ✧ Fall 2019

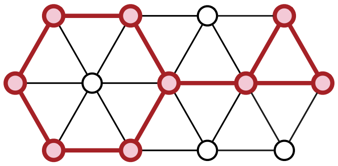
Homework 10

Due Tuesday, December 3, 2019 at 8pm

This is the last graded homework before the final exam.

This brings the total number of graded homework problems to 33,
at most 24 of which will count toward your final course grade.

1. A subset S of vertices in an undirected graph G is called ***square-free*** if, for every four distinct vertices $u, v, w, x \in S$, at least one of the four edges uv, vw, wx, xu is *absent* from G. That is, the subgraph of G induced by S has no cycles of length 4. Prove that finding the size of the largest square-free subset of vertices in a given undirected graph is NP-hard.



A square-free subset of 9 vertices, and all edges between them.
This is ***not*** the largest square-free subset in this graph.

2. Fix an alphabet $\Sigma = \{\texttt{0}, \texttt{1}\}$. Prove that the following problems are NP-hard.[1]

   (a) Given a regular expression R over the alphabet Σ, is $L(R) \neq \Sigma^*$?

   (b) Given an NFA M over the alphabet Σ, is $L(M) \neq \Sigma^*$?

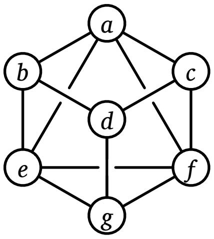
   *[Hint: Encode all the **bad** choices for some problem into a regular expression R, so that if **all** choices are bad, then $L(R) = \Sigma^*$.]*

3. ***This problem has been removed.*** — *We are deferring all discussion of undecidability until after Thanksgiving break. This problem will reappear on "Homework 11".*

[1] In fact, both of these problems are NP-hard even when $|\Sigma| = 1$, but proving that is much more difficult.

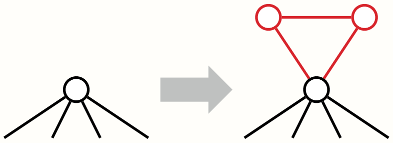
Solved Problem

4. A *double-Hamiltonian tour* in an undirected graph G is a closed walk that visits every vertex in G exactly twice. Prove that it is NP-hard to decide whether a given graph G has a double-Hamiltonian tour.



This graph contains the double-Hamiltonian tour $a\to b\to d\to g\to e\to b\to d\to c\to f\to a\to c\to f\to g\to e\to a$.

Solution: We prove the problem is NP-hard with a reduction from the standard Hamiltonian cycle problem. Let G be an arbitrary undirected graph. We construct a new graph H by attaching a small gadget to every vertex of G. Specifically, for each vertex v, we add two vertices $v^\sharp$ and $v^\flat$, along with three edges $vv^\flat$, $vv^\sharp$, and $v^\flat v^\sharp$.



A vertex in G, and the corresponding vertex gadget in H.

I claim that G has a Hamiltonian cycle if and only if H has a double-Hamiltonian tour.

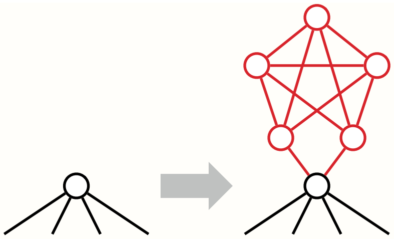
$\Longrightarrow$ Suppose G has a Hamiltonian cycle $v_1\to v_2\to\cdots\to v_n\to v_1$. We can construct a double-Hamiltonian tour of H by replacing each vertex v_i with the following walk:

$$\cdots\to v_i\to v_i^\flat\to v_i^\sharp\to v_i^\flat\to v_i^\sharp\to v_i\to\cdots$$

$\Longleftarrow$ Conversely, suppose H has a double-Hamiltonian tour D. Consider any vertex v in the original graph G; the tour D must visit v exactly twice. Those two visits split D into two closed walks, each of which visits v exactly once. Any walk from $v^\flat$ or $v^\sharp$ to any other vertex in H must pass through v. Thus, one of the two closed walks visits only the vertices v, $v^\flat$, and $v^\sharp$. Thus, if we simply remove the vertices in $H\setminus G$ from D, we obtain a closed walk in G that visits every vertex in G once.

Given any graph G, we can clearly construct the corresponding graph H in polynomial time.

With more effort, we can construct a graph H that contains a double-Hamiltonian tour ***that traverses each edge of H at most once*** if and only if G contains a Hamiltonian cycle. For each vertex v in G we attach a more complex gadget containing five vertices and eleven edges, as shown on the next page.

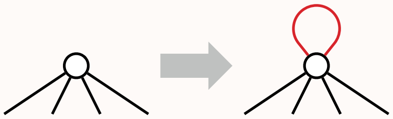


A vertex in G, and the corresponding modified vertex gadget in H.

■

Rubric: 10 points, standard polynomial-time reduction rubric. This is not the only correct solution.

Non-solution (self-loops): We attempt to prove the problem is NP-hard with a reduction from the Hamiltonian cycle problem. Let G be an arbitrary undirected graph. We construct a new graph H by attaching a self-loop every vertex of G. Given any graph G, we can clearly construct the corresponding graph H in polynomial time.

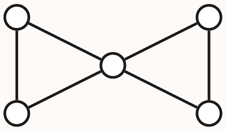


An incorrect vertex gadget.

Suppose G has a Hamiltonian cycle $v_1 \to v_2 \to \cdots \to v_n \to v_1$. We can construct a double-Hamiltonian tour of H by alternating between edges of the Hamiltonian cycle and self-loops:

$$v_1 \to v_1 \to v_2 \to v_2 \to v_3 \to \cdots \to v_n \to v_n \to v_1.$$

Unfortunately, if H has a double-Hamiltonian tour, we *cannot* conclude that G has a Hamiltonian cycle, because we cannot guarantee that a double-Hamiltonian tour in H uses *any* self-loops. The graph G shown below is a counterexample; it has a double-Hamiltonian tour (even before adding self-loops!) but no Hamiltonian cycle.



This graph has a double-Hamiltonian tour.

♣

CS/ECE 374 A ✧ Fall 2019

“Homework” 11

“Due” Monday, December 9, 2019

This homework is *not* for submission. However, undecidability questions are in scope for the final exam, so we still strongly recommend treating at least those questions as regular homework. Solutions will be released next Monday.

---

1. Let $\langle M\rangle$ denote the encoding of a Turing machine M (or if you prefer, the Python source code for the executable code M). Recall that w^R denotes the reversal of string w. Prove that the following language is undecidable.

   $$\textsc{SelfRevAccept} := \left\{\langle M\rangle \;\middle|\; M \text{ accepts the string } \langle M\rangle^R\right\}$$

   Note that Rice’s theorem does *not* apply to this language.

2. Let M be a Turing machine, let w be an arbitrary input string, and let s be an integer. We say that ***M accepts w in space s*** if, given w as input, M accesses only the first s (or fewer) cells on its tape and eventually accepts.

   (a) Prove that the following language is undecidable:

   $$\textsc{SomeSquareSpace} = \left\{\langle M\rangle \;\middle|\; M \text{ accepts at least one string } w \text{ in space } |w|^2\right\}$$

   [Hint: The only thing you need to know about Turing machines for this problem is that they consume a resource called “space”.]

   *(b) Sketch a Turing machine/algorithm that correctly decides the following language:

   $$\textsc{SquareSpace} = \left\{\langle M, w\rangle \;\middle|\; M \text{ accepts } w \text{ in space } |w|^2\right\}$$

   [Hint: This question is only for people who really want to get down in the Turing-machine weeds. Nothing like this will appear on the final exam.]

3. Consider the following language:

   $$\textsc{Picky} = \left\{\langle M\rangle \;\middle|\; \begin{array}{c} M \text{ accepts at least one input string} \\ \text{and } M \text{ rejects at least one input string} \end{array}\right\}$$

   (a) Prove that Picky is undecidable.

   (b) Sketch a Turing machine/algorithm that *accepts* Picky.

CS/ECE 374 A ✧ Fall 2019

Midterm 1 Study Questions

This is a "core dump" of potential questions for Midterm 1. This should give you a good idea of the *types* of questions that we will ask on the exam—in particular, there *will* be a series of True/False questions—but the actual exam questions may or may not appear in this list. This list intentionally includes a few questions that are too long or difficult for exam conditions; *most* of these are indicated with a *star.

Questions from Jeff's past exams are labeled with the semester they were used: ⟨⟨***S14***⟩⟩, ⟨⟨***F14***⟩⟩, ⟨⟨***F16***⟩⟩, or ⟨⟨***S18***⟩⟩. Questions from this semester's homework are labeled ⟨⟨***HW***⟩⟩. Questions from this semester's labs are labeled ⟨⟨***Lab***⟩⟩. Some unflagged questions may have been used in exams by other instructors.

How to Use These Problems

Solving every problem in this handout is ***not*** the best way to study for the exam. Memorizing the solutions to every problem in this handout is the ***absolute worst*** way to study for the exam.

What we recommend instead is to work on a *sample* of the problems. Choose one or two problems at random from each section and try to solve them from scratch under exam conditions—by yourself, in a quiet room, with a 30-minute timer, *without* your notes, *without* the internet, and if possible, even without your cheat sheet. If you're comfortable solving a few problems in a particular section, you're probably ready for that type of problem on the exam. Move on to the next section.

Discussing problems with other people (in your study groups, in the review sessions, in office hours, or on Piazza) and/or looking up old solutions can be *extremely* helpful, but ***only after*** you have (1) made a good-faith effort to solve the problem on your own, and (2) you have either a candidate solution or some idea about where you're getting stuck.

If you find yourself getting stuck on a particular type of problem, try to figure out *why* you're stuck. Do you understand the problem statement? Are you stuck on choosing the right high-level approach, are you stuck on the technical details, or are you struggling to express your ideas clearly?

Similarly, if feedback suggests that your solutions to a particular type of problem are incorrect or incomplete, try to figure out what you missed. For induction proofs: Are you sure you have the right induction hypothesis? Are your cases obviously exhaustive? For regular expressions, DFAs, NFAs, and context-free grammars: Is your solution both exclusive and exhaustive? Did you try a few positive examples *and* a few negative examples? For fooling sets: Are you imposing enough structure? Are x and y really *arbitrary* strings from F? For language transformations: Are you transforming in the right direction? Are you using non-determinism correctly? Do you understand the formal notation for DFAs and NFAs?

Remember that your goal is *not* merely to "understand" the solution to any particular problem, but to become more comfortable with solving a certain *type* of problem on your own. **"Understanding" is a trap; aim for mastery.** If you can identify specific steps that you find problematic, read more *about those steps*, focus your practice *on those steps*, and try to find helpful information *about those steps* to write on your cheat sheet. Then work on the next problem!

Induction on Strings

Give complete, formal inductive proofs for the following claims. Your proofs must reply on the formal recursive definitions of the relevant string functions, not on intuition. Recall that the concatenation • and length $|\cdot|$ functions are formally defined as follows:

$$w \bullet y := \begin{cases} y & \text{if } w = \varepsilon \\ a \cdot (x \bullet y) & \text{if } w = ax \text{ for some } a \in \Sigma \text{ and } x \in \Sigma^* \end{cases}$$

$$|w| := \begin{cases} 0 & \text{if } w = \varepsilon \\ 1 + |x| & \text{if } w = ax \text{ for some } a \in \Sigma \text{ and } x \in \Sigma^* \end{cases}$$

1.1 The ***reversal*** $\boldsymbol{w^R}$ of a string w is defined recursively as follows:

$$w^R := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ x^R \bullet a & \text{if } w = ax \text{ for some } a \in \Sigma \text{ and } x \in \Sigma^* \end{cases}$$

(a) Prove that $(w \bullet x)^R = x^R \bullet w^R$ for all strings w and x. ⟨⟨*F14*⟩⟩

(b) Prove that $(w^R)^R = w$ for every string w.

(c) Prove that $|w| = |w^R|$ for every string w.

1.2 For any string w and any non-negative integer n, let $\boldsymbol{w^n}$ denote the string obtained by concatenating n copies of w; more formally, define

$$w^n := \begin{cases} \varepsilon & \text{if } n = 0 \\ w \bullet w^{n-1} & \text{otherwise} \end{cases}$$

For example, $(\texttt{BLAH})^5 = \texttt{BLAHBLAHBLAHBLAHBLAH}$ and $\varepsilon^{374} = \varepsilon$.

(a) Prove that $w^m \bullet w^n = w^{m+n}$ for every string w and all non-negative integers n and m.

(b) Prove that $(w^m)^n = w^{mn}$ for every string w and all non-negative integers n and m.

(c) Prove that $|w^n| = n|w|$ for every string w and every integer $n \geq 0$.

(d) Prove that $(w^n)^R = (w^R)^n$ for every string w and every integer $n \geq 0$.

1.3 ⟨⟨*Lab*⟩⟩ Let $\#(a, w)$ denote the number of times symbol a appears in string w. For example, $\#(\texttt{X}, \texttt{WTF374}) = 0$ and $\#(\texttt{0}, \texttt{000010101010010100}) = 12$.

(a) Give a formal recursive definition of $\#(a, w)$.

(b) Prove that $\#(a, w \bullet z) = \#(a, w) + \#(a, z)$ for all symbols a and all strings w and z.

(c) Prove that $\#(a, w^R) = \#(a, w)$ for all symbols a and all strings w.

1.4 Consider the following pair of mutually recursive functions:

$$evens(w) := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ odds(x) & \text{if } w = ax \end{cases} \qquad odds(w) := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ a \cdot evens(x) & \text{if } w = ax \end{cases}$$

For example, $evens(\texttt{0001101}) = \texttt{010}$ and $odds(\texttt{0001101}) = \texttt{0011}$.

(a) Prove the following identity for all strings w and x:

$$evens(w \bullet x) = \begin{cases} evens(w) \bullet evens(x) & \text{if } |w| \text{ is even,} \\ evens(w) \bullet odds(x) & \text{if } |w| \text{ is odd.} \end{cases}$$

(b) State and prove a similar identity for $odds(w \bullet x)$.

(c) Prove the following identity for all strings w:

$$evens(w^R) = \begin{cases} (evens(w))^R & \text{if } |w| \text{ is odd,} \\ (odds(w))^R & \text{if } |w| \text{ is even.} \end{cases}$$

(d) Prove that $|w| = |evens(w)| + |odds(w)|$ for every string w.

1.5 The ***complement* w^c** of a string $w \in \{\texttt{0}, \texttt{1}\}^*$ is obtained from w by replacing every $\texttt{0}$ in w with a $\texttt{1}$ and vice versa. The complement function can be defined recursively as follows:

$$w^c := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ \texttt{1} \cdot x^c & \text{if } w = \texttt{0}x \\ \texttt{0} \cdot x^c & \text{if } w = \texttt{1}x \end{cases}$$

(a) Prove that $|w| = |w^c|$ for every string w.

(b) Prove that $(x \bullet y)^c = x^c \bullet y^c$ for all strings x and y.

(c) Prove that $\#(\texttt{1}, w) = \#(\texttt{0}, w^c)$ for every string w.

1.6 Consider the following recursively defined function:

$$stutter(w) := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ aa \bullet stutter(x) & \text{if } w = ax \end{cases}$$

For example, $stutter(\texttt{MISSISSIPPI}) = \texttt{MMIISSSSIISSSSIIPPPPII}$.

(a) Prove that $|stutter(w)| = 2|w|$ for every string w.

(b) Prove that $evens(stutter(w)) = w$ for every string w.

(c) Prove that $odds(stutter(w)) = w$ for every string w.

(d) Prove that w is a palindrome if and only if $stutter(w)$ is a palindrome, for every string w.

1.7 Consider the following recursive function:

$$faro(w,z) := \begin{cases} z & \text{if } w = \varepsilon \\ a \cdot faro(z,x) & \text{if } w = ax \end{cases}$$

For example, $faro(\mathtt{0011}, \mathtt{0101}) = \mathtt{00011011}$. (A "faro shuffle" splits a deck of cards into two equal piles and then perfectly interleaves them.)

(a) Prove that $|faro(x,y)| = |x| + |y|$ for all strings x and y.

(b) Prove that $faro(w,w) = stutter(w)$ for every string w.

(c) Prove that $faro(odds(w), evens(w)) = w$ for every string w.

Regular expressions

For each of the following languages over the alphabet $\{0, 1\}$, give an equivalent regular expression, and *briefly* argue why your expression is correct. (On exams, we will not ask for justifications, but you should still justify your expressions in your head.)

2.1 Every string of length at most 3. *[Hint: Don't try to be clever.]*

2.2 All strings except 010.

2.3 All strings that end with the suffix 010.

2.4 All strings that do not end with the suffix 010.

2.5 All strings that contain the substring 010.

2.6 All strings that do not contain the substring 010.

2.7 All strings that contain the subsequence 010.

2.8 All strings that do not contain the subsequence 010.

2.9 All strings containing the substring 10 or the substring 01.

2.10 ***⟨⟨F16⟩⟩*** All strings containing either the substring 10 or the substring 01, but not both.

2.11 All strings containing the subsequence 10 or the subsequence 01.

2.12 All strings containing either the subsequence 10 or the subsequence 01, but not both.

2.13 All strings containing at least two 1s and at least one 0.

2.14 All strings containing *either* at least two 1s *or* at least one 0.

2.15 ***⟨⟨lab⟩⟩*** All strings such that *in every prefix*, the number of 0s and the number of 1s differ by at most 1.

2.16 ***⟨⟨F14⟩⟩*** All strings in which every run of consecutive 0s has even length and every run of consecutive 1s has odd length.

2.17 The set of all strings in $\{0, 1\}^*$ whose length is divisible by 3.

2.18 ***⟨⟨S14⟩⟩*** The set of all strings in 0^*1^* whose length is divisible by 3.

2.19 The set of all strings in $\{0, 1\}^*$ in which the number of 1s is divisible by 3.

2.20 ***⟨⟨S18⟩⟩*** All strings in $0^*1^*0^*$ whose length is even.

2.21 ***⟨⟨S18⟩⟩*** $\left\{0^n w 1^n \;\middle|\; n \geq 1 \text{ and } w \in \Sigma^+\right\}$

2.22 All strings that end with the suffix 0000000000 (ten 0s)

2.23 All strings whose last ten symbols include an odd number of 1s.

Direct DFA construction.

Draw or formally describe a DFA that recognizes each of the following languages. Don't forget to describe the states of your DFA in English.

3.1 Every string of length at most 3.

3.2 All strings except 010.

3.3 All strings that end with the suffix 010.

3.4 All strings that do not end with the suffix 010.

3.5 All strings that contain the substring 010.

3.6 All strings that do not contain the substring 010.

3.7 All strings that contain the subsequence 010.

3.8 All strings that do not contain the subsequence 010.

3.9 The language {LONG, LUG, LEGO, LEG, LUG, LOG, LINGO} .

3.10 The language $\texttt{MOO}^* + \texttt{MEOO}^*\texttt{W}$

3.11 All strings in which every run of consecutive 0s has even length and every run of consecutive 1s has odd length. ⟨⟨*F14*⟩⟩

3.12 All strings containing the substring 10 or the substring 01.

3.13 All strings containing either the substring 10 or the substring 01, but not both.

3.14 All strings containing the subsequence 10 or the subsequence 01.

3.15 All strings containing either the subsequence 10 or the subsequence 01, but not both.

3.16 The set of all strings in $\{0,1\}^*$ whose length is divisible by 3.

3.17 ⟨⟨*S14*⟩⟩ The set of all strings in 0^*1^* whose length is divisible by 3.

3.18 The set of all strings in $\{0,1\}^*$ in which the number of 1s is divisible by 3.

3.19 All strings w such that the binary value of w^R is divisible by 5.

3.20 ⟨⟨*lab*⟩⟩ All strings such that *in every prefix*, the number of 0s and the number of 1s differ by at most 2.

3.21 ⟨⟨*S18*⟩⟩ All strings in $0^*1^*0^*$ whose length is even.

3.22 ⟨⟨*S18*⟩⟩ $\left\{0^n w 1^n \;\middle|\; n \geq 1 \text{ and } w \in \Sigma^+\right\}$

3.23 All strings that end with the suffix 0000000000 (ten 0s)

3.24 All strings whose last ten symbols include an odd number of 1s.

Fooling sets

Prove that each of the following languages is *not* regular.

4.1 The set of all strings in $\{0,1\}^*$ with more 0s than 1s. ⟨⟨*S14*⟩⟩

4.2 The set of all strings in $\{0,1\}^*$ with fewer 0s than 1s.

4.3 The set of all strings in $\{0,1\}^*$ with exactly twice as many 0s as 1s.

4.4 The set of all strings in $\{0,1\}^*$ with at least twice as many 0s as 1s.

4.5 $\left\{0^{2^n} \mid n \geq 0\right\}$ ⟨⟨*Lab*⟩⟩

4.6 $\left\{0^{F_n} \mid n \geq 0\right\}$, where F_n is the nth Fibonacci number, defined recursively as follows:

$$F_n := \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

[Hint: If $F_i + F_j$ is a Fibonacci number, then either $i = j \pm 1$ or $\min\{i,j\} \leq 2$.]

4.7 $\left\{0^{n^3} \mid n \geq 0\right\}$

4.8 $\left\{x\#y \mid x,y \in \{0,1\}^* \text{ and } \#(0,x) = \#(1,y)\right\}$

4.9 $\left\{xx^c \mid x \in \{0,1\}^*\right\}$, where x^c is the *complement* of x, obtained by replacing every 0 in x with a 1 and vice versa. For example, $0001101^c = 1110010$.

4.10 The language of properly balanced strings of parentheses, described by the context-free grammar $S \to \varepsilon \mid SS \mid (S)$. ⟨⟨*Lab*⟩⟩

4.11 $\left\{(01)^n(10)^n \mid n \geq 0\right\}$

4.12 $\left\{(01)^m(10)^n \mid n \geq m \geq 0\right\}$

4.13 $\left\{w\#x\#y \mid w,x,y \in \{0,1\}^* \text{ and } w,x,y \text{ are not all equal}\right\}$

4.14 $\left\{0^i1^j0^k \mid i = j \text{ or } j = k\right\}$ ⟨⟨*S18*⟩⟩

4.15 $\left\{0^i1^j0^k \mid 2i = k \text{ or } i = 2k\right\}$ ⟨⟨*S18*⟩⟩

Regular or Not?

For each of the following languages, either prove that the language is regular (by describing a DFA, NFA, or regular expression), or prove that the language is not regular (using a fooling set argument). Unless otherwise specified, all languages are over the alphabet $\{0, 1\}$.

5.1 ⟨⟨*F14*⟩⟩ The set of all strings in $\{0,1\}^*$ in which the substrings 01 and 10 appear the same number of times. (For example, the substrings 01 and 01 each appear three times in the string 1100001101101.)

5.2 ⟨⟨*F14*⟩⟩ The set of all strings in $\{0,1\}^*$ in which the substrings 00 and 11 appear the same number of times. (For example, the substrings 00 and 11 each appear three times in the string 1100001101101.)

5.3 ⟨⟨*F14*⟩⟩ $\{www \mid w \in \Sigma^*\}$

5.4 ⟨⟨*F14*⟩⟩ $\{wxw \mid w, x \in \Sigma^*\}$

5.5 The set of all strings in $\{0,1\}^*$ such that in every prefix, the number of 0s is greater than the number of 1s.

5.6 The set of all strings in $\{0,1\}^*$ such that in every *non-empty* prefix, the number of 0s is greater than the number of 1s.

5.7 $\{0^m 1^n \mid 0 \le m - n \le 374\}$

5.8 $\{0^m 1^n \mid 0 \le m + n \le 374\}$

5.9 The language generated by the following context-free grammar:

$$\begin{aligned} S &\to 0A1 \mid \varepsilon \\ A &\to 1S0 \mid \varepsilon \end{aligned}$$

5.10 The language generated by the following context-free grammar:

$$S \to 0S1 \mid 1S0 \mid \varepsilon$$

5.11 $\{w\#x \mid w, x \in \{0,1\}^* \text{ and no substring of } w \text{ is also a substring of } x\}$

5.12 $\{w\#x \mid w, x \in \{0,1\}^* \text{ and no \textit{non-empty} substring of } w \text{ is also a substring of } x\}$

5.13 $\{w\#x \mid w, x \in \{0,1\}^* \text{ and \textit{every} non-empty substring of } w \text{ is also a substring of } x\}$

5.14 $\{w\#x \mid w, x \in \{0,1\}^* \text{ and } w \text{ is a substring of } x\}$

5.15 $\{w\#x \mid w, x \in \{0,1\}^* \text{ and } w \text{ is a proper substring of } x\}$

5.16 $\{xy \mid \#(0,x) = \#(1,y) \text{ and } \#(1,x) = \#(0,y)\}$

5.17 $\{xy \mid \#(0,x) = \#(1,y) \text{ or } \#(1,x) = \#(0,y)\}$

5.18 $\{0^a 1^b 0^c \mid (a \le b + c \;\text{ and }\; b \le a + c) \;\text{ or }\; c \le a + b\}$ ⟨⟨*HW*⟩⟩

5.19 $\left\{\mathtt{0}^a\mathtt{1}^b\mathtt{0}^c \,\middle|\, a \le b + c \text{ and } (b \le a + c \text{ or } c \le a + b)\right\}$ ⟨⟨*HW*⟩⟩

5.20 $\left\{wxw^R \,\middle|\, w, x \in \Sigma^+\right\}$ ⟨⟨*HW*⟩⟩

5.21 $\left\{ww^Rx \,\middle|\, w, x \in \Sigma^+\right\}$ ⟨⟨*HW*⟩⟩

Product/Subset Constructions

For each of the following languages $L \subseteq \{0, 1\}^*$, formally describe a DFA $M = (Q, \{0, 1\}, s, A, \delta)$ that recognizes L. ***Do not attempt to draw the DFA.*** Instead, give a complete, precise, and self-contained description of the state set Q, the start state s, the accepting state A, and the transition function δ. Do **not** just describe several smaller DFAs and write "product construction!"

6.1 ⟨⟨*S14*⟩⟩ All strings that satisfy *all* of the following conditions:

(a) the number of 0s is even

(b) the number of 1s is divisible by 3

(c) the total length is divisible by 5

6.2 All strings that satisfy *at least one* of the following conditions: ...

6.3 All strings that satisfy *exactly one* of the following conditions: ...

6.4 All strings that satisfy *exactly two* of the following conditions: ...

6.5 All strings that satisfy *an odd number of* of the following conditions: ...

- Other possible conditions:

(a) The number of 0s in w is odd.

(b) The number of 1s in w is not divisible by 3. ⟨⟨*HW*⟩⟩

(c) The length $|w|$ is divisible by 5.

(d) The binary value of w is not divisible by 7. ⟨⟨*HW*⟩⟩

(e) The binary value of w^R is divisible by 9.

(f) w contains the substring 00.

(g) w does not contain the substring 11.

(h) w contains the substring 01 an odd number of times. ⟨⟨*HW*⟩⟩

(i) ww does not contain the substring 101.

Regular Language Transformations

Let L be an arbitrary regular language over the alphabet $\Sigma = \{\mathtt{0}, \mathtt{1}\}$. Prove that each of the following languages over $\{\mathtt{0}, \mathtt{1}\}$ is regular.

7.1 $L^c := \left\{ w^c \;\middle|\; w \in L \right\}$, where w^c is the complement of w, defined recursively as follows:

$$w^c := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ \mathtt{1} \cdot x^c & \text{if } w = \mathtt{0}x \text{ for some string } x \\ \mathtt{0} \cdot x^c & \text{if } w = \mathtt{1}x \text{ for some string } x \end{cases}$$

For example, $\mathtt{0001101}^c = \mathtt{1110010}$.

7.2 $\textsc{OneInFront}(L) := \{\mathtt{1}x \mid x \in L\}$

7.3 $\textsc{OneInBack}(L) := \{x\mathtt{1} \mid x \in L\}$

7.4 $\textsc{OnlyOnes}(L) := \left\{ \mathtt{1}^{\#(\mathtt{1},w)} \;\middle|\; w \in L \right\}$

7.5 $\textsc{PadWithZeros}(L) := \left\{ w \;\middle|\; \mathtt{1}^{\#(\mathtt{1},w)} \in L \right\}$

7.6 $\textsc{MissingFirstOne}(L) := \{w \in \Sigma^* \mid \mathtt{1}w \in L\}$

7.7 $\textsc{MissingLastOne}(L) := \{w \in \Sigma^* \mid w\mathtt{1} \in L\}$

7.8 $\textsc{Prefixes}(L) := \{x \mid xy \in L \text{ for some } y \in \Sigma^*\}$

7.9 ***⟨⟨F16⟩⟩*** $\textsc{Suffixes}(L) := \{y \mid xy \in L \text{ for some } x \in \Sigma^*\}$

7.10 $\textsc{Rotations}(L) := \{yx \mid xy \in L\}$

7.11 ***⟨⟨lab, F14⟩⟩*** $\textsc{Evens}(L) := \{\mathit{evens}(w) \mid w \in L\}$, where the functions *evens* and *odds* are recursively defined as follows:

$$\mathit{evens}(w) := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ \mathit{odds}(x) & \text{if } w = ax \end{cases} \qquad \mathit{odds}(w) := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ a \cdot \mathit{evens}(x) & \text{if } w = ax \end{cases}$$

For example, $\mathit{evens}(\mathtt{0001101}) = \mathtt{010}$ and $\mathit{odds}(\mathtt{0001101}) = \mathtt{0011}$.

7.12 ***⟨⟨lab, F14⟩⟩*** $\textsc{Evens}^{-1}(L) := \{w \mid \mathit{evens}(w) \in L\}$, where the functions *evens* and *odds* are recursively defined as above.

7.13 $\textsc{Faro}(L) := \left\{ \mathit{faro}(w, x) \;\middle|\; w, x \in L \right\}$, where the function *faro* is defined recursively as follows:

$$\mathit{faro}(w, x) := \begin{cases} x & \text{if } w = \varepsilon \\ a \cdot \mathit{faro}(x, y) & \text{if } w = ay \text{ for some } a \in \Sigma \text{ and some } y \in \Sigma^* \end{cases}$$

For example, $\mathit{faro}(\mathtt{0001101}, \mathtt{1111}) = \mathtt{0}^{\mathtt{1}}\mathtt{0}^{\mathtt{1}}\mathtt{0}^{\mathtt{1}}\mathtt{1}^{\mathtt{1}}\mathtt{101}$

7.14 $\textsc{Scramble}(L) := \left\{ \mathit{scramble}(w) \;\middle|\; w \in L \right\}$, where the function *scramble* is defined recursively as follows:

$$\mathit{scramble}(w) := \begin{cases} w & \text{if } |w| \le 1 \\ ba \cdot \mathit{scramble}(x) & \text{if } w = abx \text{ for some } a, b \in \Sigma \text{ and } x \in \Sigma^* \end{cases}$$

For example, $\mathit{scramble}(\mathtt{00\,01\,10\,1}) = \mathtt{00\,10\,01\,1}$.

7.15 ⟨⟨*S18*⟩⟩ $\textsc{OddParity}(L) := \{w \in L \mid \mathit{parity}(w) = 1\}$, where

$$\mathit{parity}(w) = \begin{cases} 0 & \text{if } \#(w, 1) \text{ is even} \\ 1 & \text{if } \#(w, 1) \text{ is odd} \end{cases}$$

7.16 $\textsc{EvenParity}(L) := \{w \in L \mid \mathit{parity}(w) = 0\}$, where $\mathit{parity}(w)$ is defined as above.

7.17 ⟨⟨*S18*⟩⟩ $\textsc{AddParityFront}(L) := \{\mathit{parity}(w) \cdot w \mid w \in L\}$, where $\mathit{parity}(w)$ is defined as above.

7.18 $\textsc{AddParityEnd}(L) := \{w \bullet \mathit{parity}(w) \mid w \in L\}$, where $\mathit{parity}(w)$ is defined as above.

7.19 ⟨⟨*S18*⟩⟩ $\textsc{StripInit0s}(L) = \left\{w \;\middle|\; 0^j w \in L \text{ for some } j \geq 0\right\}$

Context-Free Grammars

Construct context-free grammars for each of the following languages, and give a *brief* explanation of how your grammar works, including the language of each non-terminal. We explicitly do not want a formal proof of correctness.

8.1 All strings in $\{0,1\}^*$ whose length is divisible by 5.

8.2 All strings in which the substrings 01 and 01 appear the same number of times.

8.3 $\{0^n1^{2n} \mid n \geq 0\}$

8.4 $\{0^m1^n \mid n \neq 2m\}$

8.5 $\{0^i1^j0^{i+j} \mid i,j \geq 0\}$ **⟨⟨*HW*⟩⟩**

8.6 $\{0^{i+j}10^j10^i \mid i,j \geq 0\}$

8.7 $\{0^i1^j2^k \mid j \neq i+k\}$

8.8 $\left\{w\#0^{\#(0,w)} \,\middle|\, w \in \{0,1\}^*\right\}$

8.9 $\{0^i1^j2^k \mid i = j \text{ or } j = k \text{ or } i = k\}$

8.10 $\left\{0^i1^j2^k \,\middle|\, i = j \text{ or } j = k\right\}$. **⟨⟨*S18*⟩⟩**

8.11 $\{0^i1^j2^k \mid i \neq j \text{ or } j \neq k\}$

8.12 $\{0^{2i}1^{i+j}2^{2j} \mid i,j \geq 0\}$

8.13 $\left\{x\#y^R \,\middle|\, x,y \in \{0,1\}^* \text{ and } x \neq y\right\}$

8.14 All strings in $\{0,1\}^*$ that are *not* palindromes.

8.15 $\{0,1\}^* \setminus \{ww \mid w \in \{0,1\}^*\}$ **⟨⟨*lab*⟩⟩**

8.16 $\left\{0^n1^{an+b} \,\middle|\, n \geq 0\right\}$, where a and b are arbitrary natural numbers.

8.17 $\left\{0^n1^{an-b} \,\middle|\, n \geq b/a\right\}$, where a and b are arbitrary natural numbers.

True or False (sanity check)

For each statement below, check "True" if the statement is ***always*** true and "False" otherwise. Each correct answer is worth 1 point; each incorrect answer is worth —½ point; checking "I don't know" is worth ¼ point; and flipping a coin is (on average) worth ¼ point.

Read each statement *very* carefully. Some of these are deliberately subtle. On the other hand, you should not spend more than two minutes on any single statement.

Definitions

A.1 Every language is regular.

A.2 Every finite language is regular.

A.3 Every infinite language is regular. ***⟨⟨S18⟩⟩***

A.4 No infinite language is regular. ***⟨⟨S18⟩⟩***

A.5 If L is regular then L can be represented by a regular expression.

A.6 If L is not regular then L cannot be represented by a regular expression.

A.7 If L can be represented by a regular expression, then L is regular.

A.8 If L cannot be represented by a regular expression, then L is not regular.

A.9 If there is a DFA that accepts every string in L, then L is regular.

A.10 If there is a DFA that accepts every string not in L, then L is not regular.

A.11 If there is a DFA that rejects every string not in L, then L is regular.

A.12 If for every string $w \in L$ there is a DFA that accepts w, then L is regular. ***⟨⟨S14⟩⟩***

A.13 If for every string $w \notin L$ there is a DFA that rejects w, then L is regular.

A.14 If L is not regular, then for every string $w \in L$, there is a DFA that accepts w. ***⟨⟨S18⟩⟩***

A.15 If L is regular, then for every string $w \in L$, there is a DFA that rejects w. ***⟨⟨S18⟩⟩***

A.16 If some DFA recognizes L, then some NFA also recognizes L.

A.17 If some NFA recognizes L, then some DFA also recognizes L.

A.18 If some NFA with ε-transitions recognizes L, then some NFA without ε-transitions also recognizes L.

Closure Properties of Regular Languages

B.1 For all regular languages L and L', the language $L \cap L'$ is regular.

B.2 For all regular languages L and L', the language $L \cup L'$ is regular.

B.3 For all regular languages L, the language L^* is regular.

B.4 For all regular languages A, B, and C, the language $(A \cup B) \setminus C$ is regular.

B.5 For all languages $L \subseteq \Sigma^*$, if L is regular, then $\Sigma^* \setminus L$ is regular.

B.6 For all languages $L \subseteq \Sigma^*$, if L is regular, then $\Sigma^* \setminus L$ is not regular.

B.7 For all languages $L \subseteq \Sigma^*$, if L is not regular, then $\Sigma^* \setminus L$ is regular.

B.8 For all languages $L \subseteq \Sigma^*$, if L is not regular, then $\Sigma^* \setminus L$ is not regular.

B.9 ⟨⟨*S14*⟩⟩ For all languages L and L', the language $L \cap L'$ is regular.

B.10 ⟨⟨*F14*⟩⟩ For all languages L and L', the language $L \cup L'$ is regular.

B.11 For all languages L, the language L^* is regular. ⟨⟨*F14, F16*⟩⟩

B.12 For all languages L, if L^* is regular, then L is regular.

B.13 For all languages A, B, and C, the language $(A \cup B) \setminus C$ is regular.

B.14 For all languages L, if L is finite, then L is regular.

B.15 For all languages L and L', if L and L' are finite, then $L \cup L'$ is regular.

B.16 For all languages L and L', if L and L' are finite, then $L \cap L'$ is regular.

B.17 For all languages $L \subseteq \Sigma^*$, if L contains infinitely many strings in Σ^*, then L is not regular.

B.18 ⟨⟨*S14*⟩⟩ For all languages $L \subseteq \Sigma^*$, if L contains all but a finite number of strings of Σ^*, then L is regular.

B.19 For all languages $L \subseteq \{0, 1\}^*$, if L contains a finite number of strings in 0^*, then L is regular.

B.20 For all languages $L \subseteq \{0, 1\}^*$, if L contains all but a finite number of strings in 0^*, then L is regular.

B.21 If L and L' are not regular, then $L \cap L'$ is not regular.

B.22 If L and L' are not regular, then $L \cup L'$ is not regular.

B.23 If L is regular and $L \cup L'$ is regular, then L' is regular. ⟨⟨*S14*⟩⟩

B.24 If L is regular and $L \cup L'$ is not regular, then L' is not regular. ⟨⟨*S14*⟩⟩

B.25 If L is not regular and $L \cup L'$ is regular, then L' is regular.

B.26 If L is regular and $L \cap L'$ is regular, then L' is regular.

B.27 If L is regular and $L \cap L'$ is not regular, then L' is not regular.

B.28 If L is regular and L' is finite, then $L \cup L'$ is regular. ⟨⟨*S14*⟩⟩

B.29 If L is regular and L' is finite, then $L \cap L'$ is regular.

B.30 If L is regular and $L \cap L'$ is finite, then L' is regular.

B.31 If L is regular and $L \cap L' = \varnothing$, then L' is regular. ⟨⟨*S18*⟩⟩

B.32 If L is regular and $L \cap L' = \varnothing$, then L' is not regular. ⟨⟨*S18*⟩⟩

B.33 If L is not regular and $L \cap L' = \varnothing$, then L' is regular. ⟨⟨*F16*⟩⟩

B.34 If L is regular and L' is not regular, then $L \cap L' = \varnothing$.

B.35 If $L \subseteq L'$ and L is regular, then L' is regular.

B.36 If $L \subseteq L'$ and L' is regular, then L is regular. ⟨⟨*F14*⟩⟩

B.37 If $L \subseteq L'$ and L is not regular, then L' is not regular.

B.38 If $L \subseteq L'$ and L' is not regular, then L is not regular. ⟨⟨*F14*⟩⟩

B.39 For every regular language L, the language $\{\mathtt{0}^{|w|} \mid w \in L\}$ is also regular. ⟨⟨*S18*⟩⟩

B.40 For every non-regular language L, the language $\{\mathtt{0}^{|w|} \mid w \in L\}$ is also non-regular. ⟨⟨*S18*⟩⟩

B.41 For all languages $L \subseteq \Sigma^*$, if L cannot be described by a regular expression, then some DFA accepts $\Sigma^* \setminus L$.

B.42 For all languages $L \subseteq \Sigma^*$, if no DFA accepts L, then the complement $\Sigma^* \setminus L$ can be described by a regular expression.

B.43 For all languages $L \subseteq \Sigma^*$, if no DFA accepts L, then the complement $\Sigma^* \setminus L$ cannot be described by a regular expression.

B.44 For all languages $L \subseteq \Sigma^*$, if L is recognized by a DFA, then $\Sigma^* \setminus L$ can be described by a regular expression. ⟨⟨*F16*⟩⟩

Properties of Context-free Languages

C.1 If L cannot be recognized by a DFA, then L is context-free.

C.2 If L cannot be recognized by a DFA, then L is not context-free.

C.3 If L can be recognized by a DFA, then L is context-free.

C.4 If L can be recognized by a DFA, then L is not context-free.

C.5 If L is not context-free, then L is regular.

C.6 If L is not context-free, then $\Sigma^* \setminus L$ is regular.

C.7 If L is not context-free, then L is not regular.

C.8 If L is not context-free, then $\Sigma^* \setminus L$ is not regular.

C.9 Every finite language is context-free.

C.10 Every context-free language is regular. ⟨⟨*F14*⟩⟩

C.11 Every regular language is context-free.

C.12 Every infinite language is context-free.

C.13 Every non-context-free language is non-regular. ⟨⟨*F16*⟩⟩

C.14 For all context-free languages L and L', the language $L \bullet L'$ is also context-free. ⟨⟨*F16*⟩⟩

C.15 For every context-free language L, the language L^* is also context-free.

C.16 For all context-free languages A, B, and C, the language $(A \cup B)^* \bullet C$ is also context-free.

C.17 For every language L, the language L^* is context-free.

C.18 For every language L, if L^* is context-free then L is context-free.

C.19 For every context-free language L, the language $\left\{\mathtt{0}^{|w|} \mid w \in L\right\}$ is also context-free. ⟨⟨*S18*⟩⟩

Equivalence Classes. Recall that for any language $L \subset \Sigma^*$, two strings $x, y \in \Sigma^*$ are equivalent with respect to L if and only if, for every string $z \in \Sigma^*$, either both xz and yz are in L, or neither xz nor yz is in L—or more concisely, if x and y have no distinguishing suffix with respect to L. We denote this equivalence by $x \equiv_L y$.

D.1 For all languages L, if L is regular, then $\equiv_L$ has finitely many equivalence classes.

D.2 For all languages L, if L is not regular, then $\equiv_L$ has infinitely many equivalence classes. ⟨⟨*S14*⟩⟩

D.3 For all languages L, if $\equiv_L$ has finitely many equivalence classes, then L is regular.

D.4 For all languages L, if $\equiv_L$ has infinitely many equivalence classes, then L is not regular.

D.5 For all regular languages L, each equivalence class of $\equiv_L$ is a regular language.

D.6 For all languages L, each equivalence class of $\equiv_L$ is a regular language.

Fooling Sets

E.1 If L has an infinite fooling set, then L is not regular.

E.2 If L has an finite fooling set, then L is regular.

E.3 If L does not have an infinite fooling set, then L is regular.

E.4 If L is not regular, then L has an infinite fooling set.

E.5 If L is regular, then L has no infinite fooling set.

E.6 If L is not regular, then L has no finite fooling set. ⟨⟨*F14, F16*⟩⟩

E.7 If L is context-free and L has a finite fooling set, then L is regular. ⟨⟨*S18*⟩⟩

E.8 If L is context-free and L has a finite fooling set, then L is not regular. ⟨⟨*S18*⟩⟩

E.9 If L is context-free and L has an infinite fooling set, then L is not regular.

E.10 If L is not context-free and L has no infinite fooling set, then L is not regular. *[Hint: Careful!]*

Specific Languages (Gut Check). Do *not* construct complete DFAs, NFAs, regular expressions, or fooling-set arguments for these languages. You don't have time.

F.1 $\{0^i 1^j 0^k \mid i + j - k = 374\}$ is regular. ⟨⟨*S14*⟩⟩

F.2 $\{0^i 1^j 0^k \mid i + j - k \geq 374\}$ is regular. ⟨⟨*S18*⟩⟩

F.3 $\{0^i 1^j 0^k \mid i + j + k = 374\}$ is regular.

F.4 $\{0^i 1^j 0^k \mid i + j + k \geq 374\}$ is regular. ⟨⟨*S18*⟩⟩

F.5 $\{0^i 1^j \mid i < 374 < j\}$ is regular. ⟨⟨*S14*⟩⟩

F.6 $\left\{0^i 1^j \;\middle|\; 0 \leq i + j \leq 374\right\}$ is regular. ⟨⟨*F14*⟩⟩

F.7 $\left\{0^i 1^j \;\middle|\; 0 \leq i - j \leq 374\right\}$ is regular. ⟨⟨*F14*⟩⟩

F.8 $\{0^i 1^j \mid i, j \geq 0\}$ is regular. ⟨⟨*F16*⟩⟩

F.9 $\{0^i 1^j \mid (i - j)$ is divisible by 374$\}$ is regular. ⟨⟨*S14*⟩⟩

F.10 $\{0^i 1^j \mid (i + j)$ is divisible by 374$\}$ is regular.

F.11 $\left\{0^{n^2} \;\middle|\; n \geq 0\right\}$ is regular.

F.12 $\left\{0^{37n+4} \;\middle|\; n \geq 0\right\}$ is regular.

F.13 $\left\{0^n 1 0^n \;\middle|\; n \geq 0\right\}$ is regular.

F.14 $\left\{0^m 1 0^n \;\middle|\; m \geq 0 \text{ and } n \geq 0\right\}$ is regular.

F.15 $\{w \in \{0, 1\}^* \mid |w|$ is divisible by 374$\}$ is regular.

F.16 $\{w \in \{0, 1\}^* \mid w$ represents a integer divisible by 374 in binary$\}$ is regular.

F.17 $\{w \in \{0, 1\}^* \mid w$ represents a integer divisible by 374 in base 473$\}$ is regular.

F.18 $\left\{w \in \{0, 1\}^* \;\middle|\; |\#(0, w) - \#(1, w)| < 374\right\}$ is regular.

F.19 $\left\{w \in \{0, 1\}^* \;\middle|\; |\#(0, x) - \#(1, x)| < 374 \text{ for every prefix } x \text{ of } w\right\}$ is regular.

F.20 $\left\{w \in \{0, 1\}^* \;\middle|\; |\#(0, x) - \#(1, x)| < 374 \text{ for every substring } x \text{ of } w\right\}$ is regular.

F.21 $\left\{w 0^{\#(0,w)} \;\middle|\; w \in \{0, 1\}^*\right\}$ is regular.

F.22 $\left\{w 0^{\#(0,w) \bmod 374} \;\middle|\; w \in \{0, 1\}^*\right\}$ is regular.

Automata Transformations

G.1 Let $M = (\Sigma, Q, s, A, \delta)$ and $M' = (\Sigma, Q, s, Q \setminus A, \delta)$ be arbitrary **DFAs** with identical alphabets, states, starting states, and transition functions, but with complementary accepting states. Then $L(M) \cap L(M') = \varnothing$. *⟨⟨F16⟩⟩*

G.2 Let $M = (\Sigma, Q, s, A, \delta)$ and $M' = (\Sigma, Q, s, Q \setminus A, \delta)$ be arbitrary **NFAs** with identical alphabets, states, starting states, and transition functions, but with complementary accepting states. Then $L(M) \cap L(M') = \varnothing$. *⟨⟨F16⟩⟩*

G.3 Let M be a **DFA** over the alphabet Σ. Let M' be identical to M, except that accepting states in M are non-accepting in M' and vice versa. Each string in Σ^* is accepted by exactly one of M and M'.

G.4 Let M be an **NFA** over the alphabet Σ. Let M' be identical to M, except that accepting states in M are non-accepting in M' and vice versa. Each string in Σ^* is accepted by exactly one of M and M'.

G.5 Let $M = (\Sigma, Q, s, A, \delta)$ be an arbitrary **DFA** over the alphabet $\Sigma = \{0, 1\}$, and let $M' = (\Sigma, Q, s, A, \delta')$ be the DFA obtained from M by changing every 0-transition into a 1-transition and vice versa. More formally, M and M' have the same states, input alphabet, starting state, and accepting states, but $\delta'(q, 0) = \delta(q, 1)$ and $\delta'(q, 1) = \delta(q, 0)$ for every state q. Then $L(M) \cup L(M') = \{0, 1\}^*$. *⟨⟨S18⟩⟩*

G.6 Let $M = (\Sigma, Q, s, A, \delta)$ be an arbitrary **DFA** over the alphabet $\Sigma = \{0, 1\}$, and let $M' = (\Sigma, Q, s, A, \delta')$ be the DFA obtained from M by changing every 0-transition into a 1-transition and vice versa, as in the previous question. Then $L(M) \cap L(M') = \varnothing$. *⟨⟨S18⟩⟩*

G.7 Let $M = (\Sigma, Q, s, A, \delta)$ be an arbitrary **NFA** over the alphabet $\Sigma = \{0, 1\}$, and let $M' = (\Sigma, Q, s, A, \delta')$ be the NFA obtained from M by changing every 0-transition into a 1-transition and vice versa, as in the two previous questions. Then $L(M) \cap L(M') = \varnothing$.

G.8 Let $M = (\Sigma, Q, s, A, \delta)$ be an arbitrary NFA, and $M' = (\Sigma, Q', s, A', \delta')$ be any NFA obtained from M by deleting some subset of the states. More formally, $Q' \subseteq Q$ and $A' = A \cap Q'$ and $\delta'(q, a) = \delta(q, a) \cap Q'$ for every state $q \in Q'$. Then $L(M') \subseteq L(M)$. *⟨⟨S18⟩⟩*

G.9 If a language L is recognized by a DFA with n states, then the complementary language $\Sigma^* \setminus L$ is recognized by a DFA with at most $n + 1$ states.

G.10 If a language L is recognized by an NFA with n states, then the complementary language $\Sigma^* \setminus L$ is recognized by a NFA with at most $n + 1$ states.

G.11 If a language L is recognized by a DFA with n states, then L^* is recognized by a DFA with at most $n + 1$ states.

G.12 If a language L is recognized by an NFA with n states, then L^* is recognized by a NFA with at most $n + 1$ states.

Language Transformations

H.1 For every regular language L, the language $\left\{w^R \;\middle|\; w \in L\right\}$ is also regular.

H.2 For every language L, if the language $\left\{w^R \;\middle|\; w \in L\right\}$ is regular, then L is also regular. ***⟨⟨F14⟩⟩***

H.3 For every language L, if the language $\left\{w^R \;\middle|\; w \in L\right\}$ is not regular, then L is also not regular. ***⟨⟨F14⟩⟩***

H.4 For every regular language L, the language $\left\{w \;\middle|\; ww^R \in L\right\}$ is also regular.

H.5 For every regular language L, the language $\left\{ww^R \;\middle|\; w \in L\right\}$ is also regular.

H.6 For every language L, if the language $\left\{w \;\middle|\; ww^R \in L\right\}$ is regular, then L is also regular. *[Hint: Consider the language $L = \{0^n1^n \mid n \ge 0\}$.]*

H.7 For every regular language L, the language $\left\{0^{|w|} \;\middle|\; w \in L\right\}$ is also regular.

H.8 For every language L, if the language $\left\{0^{|w|} \;\middle|\; w \in L\right\}$ is regular, then L is also regular.

H.9 For every context-free language L, the language $\left\{w^R \;\middle|\; w \in L\right\}$ is also context-free.

CS/ECE 374 A ✧ Fall 2019

Fake Midterm 1

September 30, 2019

Real name:	
NetID:	

Gradescope name:	
Gradescope email:	

- ***Don't panic!***
- If you brought anything except your writing implements, your **hand-written** double-sided 8½" × 11" cheat sheet, and your university ID, please put it away for the duration of the exam. In particular, please turn off and put away *all* medically unnecessary electronic devices.
- Please clearly print your real name, your university NetID, your Gradescope name, and your Gradescope email address in the boxes above. However, if you are using your real name and your university email address on Gradescope, you do ***not*** need to write everything twice. **We will not scan this page into Gradescope.**
- Please also print **only the name you are using on Gradescope** at the top of every page of the answer booklet, except this cover page. These are the pages we will scan into Gradescope.
- Please do not write outside the black boxes on each page; these indicate the area of the page that the scanner can actually see.
- If you run out of space for an answer, feel free to use the scratch pages at the back of the answer booklet, but **please clearly indicate where we should look**.
- Proofs are required for full credit if and only if we explicitly ask for them, using the word ***prove*** in bold italics.
- Please return ***all*** paper with your answer booklet: your question sheet, your cheat sheet, and all scratch paper.

CS/ECE 374 A ✧ Fall 2019 **Fake Midterm 1 Problem 1**	Gradescope name:

For each statement below, check "True" if the statement is ***always*** true and "False" otherwise. Each correct answer is worth +1 point; each incorrect answer is worth −½ point; checking "I don't know" is worth +¼ point; and flipping a coin is (on average) worth +¼ point.

□ Yes □ No □ IDK Every integer in the empty set is prime.

□ Yes □ No □ IDK The language $\{\mathtt{0}^m\mathtt{1}^n \mid m+n \leq 374\}$ is regular.

□ Yes □ No □ IDK The language $\{\mathtt{0}^m\mathtt{1}^n \mid m-n \leq 374\}$ is regular.

□ Yes □ No □ IDK For all languages L, the language L^* is regular.

□ Yes □ No □ IDK For all languages L, the language L^* is infinite.

□ Yes □ No □ IDK For all languages $L \subset \Sigma^*$, if L can be represented by a regular expression, then $\Sigma^* \setminus L$ is recognized by a DFA.

□ Yes □ No □ IDK For all languages L and L', if $L \cap L' = \varnothing$ and L' is not regular, then L is regular.

□ Yes □ No □ IDK Let $M = (\Sigma, Q, s, A, \delta)$ and $M' = (\Sigma, Q, s, Q \setminus A, \delta)$ be arbitrary **DFA**s with identical alphabets, states, starting states, and transition functions, but with complementary accepting states. Then $L(M) \cap L(M') = \varnothing$.

□ Yes □ No □ IDK Let $M = (\Sigma, Q, s, A, \delta)$ and $M' = (\Sigma, Q, s, Q \setminus A, \delta)$ be arbitrary **NFA**s with identical alphabets, states, starting states, and transition functions, but with complementary accepting states. Then $L(M) \cap L(M') = \varnothing$.

□ Yes □ No □ IDK For all context-free language L, the language L^* is also context-free.

CS/ECE 374 A ✧ Fall 2019 **Fake Midterm 1 Problem 2**	Gradescope name:

For each of the following languages over the alphabet $\Sigma = \{0, 1\}$, either ***prove*** that the language is regular or ***prove*** that the language is not regular. ***Exactly one of these two languages is regular.*** Both of these languages contain the string 00110100000110100.

1. $\left\{0^n w 0^n \mid w \in \Sigma^+ \text{ and } n > 0\right\}$

2. $\left\{w 0^n w \mid w \in \Sigma^+ \text{ and } n > 0\right\}$

CS/ECE 374 A ✧ Fall 2019 **Fake Midterm 1 Problem 3**	Gradescope name:

The *parity* of a bit-string w is $\mathtt{0}$ if w has an even number of $\mathtt{1}$s, and $\mathtt{1}$ if w has an odd number of $\mathtt{1}$s. For example:

$$parity(\varepsilon) = \mathtt{0} \qquad parity(\mathtt{0010100}) = \mathtt{0} \qquad parity(\mathtt{00101110100}) = \mathtt{1}$$

(a) Give a *self-contained*, formal, recursive definition of the *parity* function. (In particular, do ***not*** refer to # or other functions defined in class.)

(b) Let L be an arbitrary regular language. Prove that the language $OddParity(L) := \{w \in L \mid parity(w) = \mathtt{1}\}$ is also regular.

(c) Let L be an arbitrary regular language. Prove that the language $AddParity(L) := \{parity(w) \cdot w \mid w \in L\}$ is also regular.

[Hint: Yes, you have enough room.]

CS/ECE 374 A ✧ Fall 2019 Fake Midterm 1 Problem 4	Gradescope name:

For each of the following languages L, give a regular expression that represents L ***and*** describe a DFA that recognizes L. You do ***not*** need to prove that your answers are correct.

(a) All strings in $(0+1)^*$ that do not contain the substring 0110.

(b) All strings in 0^*10^* whose length is a multiple of 3.

CS/ECE 374 A ✧ Fall 2019 **Fake Midterm 1 Problem 5**	Gradescope name:

Consider the language L of all strings in $\{\mathtt{0},\mathtt{1}\}^*$ in which the number of `0`s is even, the number of `1`s is divisible by 3, and the total number of symbols is divisible by 5. For example, the strings `01011` and `0000000000` are in L, but the strings `01001` and `10101010` are not.

Formally describe a DFA $M = (Q, s, A, \delta)$ over the alphabet $\Sigma = \{\mathtt{0},\mathtt{1}\}$ that recognizes L. ***Do not attempt to draw the DFA. Do not use the phrase "product construction".*** Instead, formally and explicitly specify each of the the components Q, s, A, and δ.

Write your answers in the separate answer booklet.

Please return this question sheet and your cheat sheet with your answers.

1. For each statement below, check "Yes" if the statement is ***always*** true and "No" otherwise. Each correct answer is worth +1 point; each incorrect answer is worth −½ point; checking "I don't know" is worth +¼ point; and flipping a coin is (on average) worth +¼ point. You do ***not*** need to prove your answer is correct.

   Read each statement *very* carefully. Some of these are deliberately subtle.

   (a) If $2+2=5$, then zero is odd.

   (b) Language L is regular if and only if there is a DFA that accepts every string in L.

   (c) Two languages L and L' are regular if and only if $L \cup L'$ is regular.

   (d) For every language L, if L^* is empty, then L is empty.

   (e) Every regular language is recognized by a DFA with exactly one accepting state.

   (f) If L has a fooling set of size 374, then L is regular.

   (g) The language $\left\{\mathtt{0}^{374n} \mid n \geq 374\right\}$ is regular.

   (h) The language $\left\{\mathtt{0}^{37n}\mathtt{1}^{4n} \mid n \geq 374\right\}$ is regular.

   (i) The language $\left\{\mathtt{0}^{3n}\mathtt{1}^{74n} \mid n \leq 374\right\}$ is regular.

   (j) Every language is either regular or context-free.

2. For any string $w \in \{\mathtt{0},\mathtt{1}\}^*$, let $slash(w)$ be the string in $\{\mathtt{0},\mathtt{1},\mathtt{/}\}^*$ obtained from w by inserting a new symbol $\mathtt{/}$ between any two consecutive appearances of the same symbol. For example:

$$slash(\varepsilon) = \varepsilon$$
$$slash(\mathtt{10101}) = \mathtt{10101}$$
$$slash(\mathtt{001010111}) = \mathtt{0/010101/1/1}$$

   For any language $L \subseteq \{\mathtt{0},\mathtt{1}\}^*$, let $slash(L) = \{slash(w) \mid w \in L\}$.

   (a) Draw or describe a DFA that accepts the language $slash(\{\mathtt{0},\mathtt{1}\}^*)$.

   (b) Give a regular expression for the language $slash(\{\mathtt{0},\mathtt{1}\}^*)$.

   (c) ***Prove*** that for any regular language L, the language $slash(L)$ is also regular.

   (You do not need to justify your answers to parts (a) and (b).)

3. Let L be the language $\{0^a 1^b 0^c \mid 2a = b + c\}$.

   (a) ***Prove*** that L is not a regular language.

   (b) Describe a context-free grammar for L. (You do not need to justify your answer.)

4. For each of the following languages L, give a regular expression that represents L ***and*** draw or describe a DFA that recognizes L. You do not need to justify your answers.

   (a) All strings in $\{0, 1\}^*$ that do not contain either 100 or 011 as a substring

   (b) All strings in $\{0, 1, 2\}^*$ that do not contain either 01 or 12 or 20 as a substring

5. For any string $w \in \{0, 1\}^*$, let *stupefy*(w) denote the string obtained from w by deleting the first 1 (if any) and replacing each remaining 1 with a 0. For example:

$$
\begin{aligned}
\textit{stupefy}(\varepsilon) &= \varepsilon \\
\textit{stupefy}(000) &= 000 \\
\textit{stupefy}(00100) &= 0000 \\
\textit{stupefy}(111111) &= 00000 \\
\textit{stupefy}(0100001101) &= 000000000
\end{aligned}
$$

   Let L be an arbitrary regular language.

   (a) ***Prove*** that the language $\{\textit{stupefy}(w) \mid w \in L\}$ is regular.

   (b) ***Prove*** that the language $\{w \in \{0, 1\}^* \mid \textit{stupefy}(w) \in L\}$ is regular.

Write your answers in the separate answer booklet.

Please return this question sheet and your cheat sheet with your answers.

1. For each statement below, check "Yes" if the statement is ***always*** true and "No" otherwise. Each correct answer is worth $+1$ point; each incorrect answer is worth $-1/2$ point; checking "I don't know" is worth $+1/4$ point; and flipping a coin is (on average) worth $+1/4$ point. You do ***not*** need to prove your answer is correct.

   Read each statement *very* carefully. Some of these are deliberately subtle.

   (a) If zero is odd, then $2+2=5$.

   (b) For every language L, and for every string $w \in L$, there is a DFA that accepts w.

   (c) Two languages L and L' are regular if and only if $L \cap L'$ is regular.

   (d) For every language L, the language L^* is non-empty.

   (e) Every regular language is recognized by an NFA with exactly 374 accepting states.

   (f) If L does not have a fooling set of size 374, then L is regular.

   (g) The language $\left\{0^{374n} \mid n \geq 374\right\}$ is regular.

   (h) The language $\left\{0^{37n}1^{4n} \mid n \geq 374\right\}$ is regular.

   (i) The language $\left\{0^{3n}1^{74n} \mid n \leq 374\right\}$ is regular.

   (j) The empty language is context-free.

2. For any string $w \in \{0,1\}^*$, let $slash(w)$ be the string in $\{0,1,/\}^*$ obtained from w by inserting a new symbol / between any two consecutive symbols that are *not* equal. For example:

$$slash(\varepsilon) = \varepsilon$$
$$slash(00000) = 00000$$
$$slash(000110111) = 000/11/0/111$$

   For any language $L \subseteq \{0,1\}^*$, let $slash(L) = \{slash(w) \mid w \in L\}$.

   (a) Draw or describe a DFA that accepts the language $slash(\{0,1\}^*)$.

   (b) Give a regular expression for the language $slash(\{0,1\}^*)$.

   (c) ***Prove*** that for any regular language L, the language $slash(L)$ is also regular.

   (You do not need to justify your answers to parts (a) and (b).)

3. Let L be the language $\left\{ \mathtt{0}^a \mathtt{1}^b \mathtt{0}^c \;\middle|\; a + b = 2c \right\}$

   (a) ***Prove*** that L is not a regular language.

   (b) Describe a context-free grammar for L. (You do not need to justify your answer.)

4. For each of the following languages L, give a regular expression that represents L ***and*** draw or describe a DFA that recognizes L. You do not need to justify your answers.

   (a) All strings in $\{\mathtt{0}, \mathtt{1}\}^*$ that do not contain either `001` or `110` as a substring

   (b) All strings in $\{\mathtt{0}, \mathtt{1}, \mathtt{2}\}^*$ that do not contain either `01` or `12` as a substring

5. For any string $w \in \{\mathtt{0}, \mathtt{1}\}^*$, let *obliviate*$(w)$ denote the string obtained from w by removing every `1`. For example:

$$\begin{aligned} \mathit{obliviate}(\varepsilon) &= \varepsilon \\ \mathit{obliviate}(\mathtt{000000}) &= \mathtt{000000} \\ \mathit{obliviate}(\mathtt{111111}) &= \varepsilon \\ \mathit{obliviate}(\mathtt{0100001101}) &= \mathtt{000000} \end{aligned}$$

   Let L be an arbitrary regular language.

   (a) ***Prove*** that the language $\{\mathit{obliviate}(w) \mid w \in L\}$ is regular.

   (b) ***Prove*** that the language $\{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid \mathit{obliviate}(w) \in L\}$ is regular.

CS/ECE 374 A ✧ Fall 2019

Midterm 2 Study Questions

This is a "core dump" of potential questions for Midterm 1. This should give you a good idea of the *types* of questions that we will ask on the exam—in particular, there *will* be a series of True/False questions—but the actual exam questions may or may not appear in this list. This list intentionally includes a few questions that are too long or difficult for exam conditions; *most* of these are indicated with a *star.

Questions from Jeff's past exams are labeled with the semester they were used: ⟨⟨***S14***⟩⟩, ⟨⟨***F14***⟩⟩, ⟨⟨***F16***⟩⟩, or ⟨⟨***S18***⟩⟩. Questions from this semester's homework are labeled ⟨⟨***HW***⟩⟩. Questions from this semester's labs are labeled ⟨⟨***Lab***⟩⟩. Some unflagged questions may have been used in exams by other instructors.

How to Use These Problems

Solving every problem in this handout is ***not*** the best way to study for the exam. Memorizing the solutions to every problem in this handout is the ***absolute worst*** way to study for the exam.

What we recommend instead is to work on a *sample* of the problems. Choose one or two problems at random from each section and try to solve them from scratch under exam conditions—by yourself, in a quiet room, with a 30-minute timer, *without* your notes, *without* the internet, and if possible, even without your cheat sheet. If you're comfortable solving a few problems in a particular section, you're probably ready for that type of problem on the exam. Move on to the next section.

Discussing problems with other people (in your study groups, in the review sessions, in office hours, or on Piazza) and/or looking up old solutions can be *extremely* helpful, but ***only after*** you have (1) made a good-faith effort to solve the problem on your own, and (2) you have either a candidate solution or some idea about where you're getting stuck.

If you find yourself getting stuck on a particular type of problem, try to figure out *why* you're stuck. Do you understand the problem statement? Have you tried several example inputs to see what the correct output *should* be? Are you stuck on choosing the right high-level approach, are you stuck on the details, or are you struggling to express your ideas clearly?

Similarly, if feedback suggests that your solutions to a particular type of problem are incorrect or incomplete, try to figure out what you missed. For recursion/dynamic programming: Are you solving the right recursive generalization of the stated problem? Are you having trouble writing a specification of the function, as opposed to a description of the algorithm? Are you struggling to find a good evaluation order? Are you trying to use a greedy algorithm? *[Hint: Don't.]* For graph algorithms: Are you aiming for the right problem? Are you having trouble figuring out the interesting states of the problem (otherwise known as vertices) and the transitions between them (otherwise known as edges)? Are you trying to modify a standard algorithm to fit the problem instead of modifying the input to fit some standard algorithm?

Remember that your goal is *not* merely to "understand" the solution to any particular problem, but to become more comfortable with solving a certain *type* of problem on your own. **"Understanding" is a trap; aim for mastery.** If you can identify specific steps that you find problematic, read more *about those steps*, focus your practice *on those steps*, and try to find helpful information *about those steps* to write on your cheat sheet. Then work on the next problem!

Recursion and Dynamic Programming

Elementary Recursion/Divide and Conquer

1. ⟨⟨*Lab*⟩⟩

   (a) Suppose $A[1\,..\,n]$ is an array of n distinct integers, sorted so that $A[1] < A[2] < \cdots < A[n]$. Each integer $A[i]$ could be positive, negative, or zero. Describe a fast algorithm that either computes an index i such that $A[i] = i$ or correctly reports that no such index exists..

   (b) Now suppose $A[1\,..\,n]$ is a sorted array of n distinct ***positive*** integers. Describe an even faster algorithm that either computes an index i such that $A[i] = i$ or correctly reports that no such index exists. *[Hint: This is* **really** *easy.]*

2. ⟨⟨*Lab*⟩⟩ Suppose we are given an array $A[1\,..\,n]$ such that $A[1] \geq A[2]$ and $A[n-1] \leq A[n]$. We say that an element $A[x]$ is a ***local minimum*** if both $A[x-1] \geq A[x]$ and $A[x] \leq A[x+1]$. For example, there are exactly six local minima in the following array:

9	7	7	2	1	3	7	5	4	7	3	3	4	8	6	9
	▲			▲				▲		▲	▲			▲	

   Describe and analyze a fast algorithm that returns the index of one local minimum. For example, given the array above, your algorithm could return the integer 5, because $A[5]$ is a local minimum. *[Hint: With the given boundary conditions, any array* ***must*** *contain at least one local minimum. Why?]*

3. ⟨⟨*Lab*⟩⟩ Suppose you are given two sorted arrays $A[1\,..\,n]$ and $B[1\,..\,n]$ containing distinct integers. Describe a fast algorithm to find the median (meaning the nth smallest element) of the union $A \cup B$. For example, given the input

$$A[1\,..\,8] = [0, 1, 6, 9, 12, 13, 18, 20] \qquad B[1\,..\,8] = [2, 4, 5, 8, 17, 19, 21, 23]$$

   your algorithm should return the integer 9. *[Hint: What can you learn by comparing one element of A with one element of B?]*

4. ⟨⟨*F14, S14*⟩⟩ An array $A[0\,..\,n-1]$ of n distinct numbers is ***bitonic*** if there are unique indices i and j such that $A[(i-1) \bmod n] < A[i] > A[(i+1) \bmod n]$ and $A[(j-1) \bmod n] > A[j] < A[(j+1) \bmod n]$. In other words, a bitonic sequence either consists of an increasing sequence followed by a decreasing sequence, or can be circularly shifted to become so. For example,

4	6	9	8	7	5	1	2	3	is bitonic, but
3	6	9	8	7	5	1	2	4	is *not* bitonic.

   Describe and analyze an algorithm to find the index of the *smallest* element in a given bitonic array $A[0\,..\,n-1]$ in $O(\log n)$ time. You may assume that the numbers in the input array are distinct. For example, given the first array above, your algorithm should return 6, because $A[6] = 1$ is the smallest element in that array.

5. ⟨⟨*F16*⟩⟩ Suppose you are given a sorted array $A[1\,..\,n]$ of distinct numbers that has been *rotated* k steps, for some ***unknown*** integer k between 1 and $n-1$. That is, the prefix $A[1\,..\,k]$ is sorted in increasing order, the suffix $A[k+1\,..\,n]$ is sorted in increasing order, and $A[n] < A[1]$. For example, you might be given the following 16-element array (where $k = 10$):

9	13	16	18	19	23	28	31	37	42	−4	0	2	5	7	8

Describe and analyze an efficient algorithm to determine if the given array contains a given number x. The input to your algorithm is the array $A[1\,..\,n]$ and the number x; your algorithm is ***not*** given the integer k.

6. ⟨⟨*F16*⟩⟩ Suppose you are given two unsorted arrays $A[1\,..\,n]$ and $B[1\,..\,n]$ containing $2n$ distinct integers, such that $A[1] < B[1]$ and $A[n] > B[n]$. Describe and analyze an efficient algorithm to compute an index i such that $A[i] < B[i]$ and $A[i+1] > B[i+1]$. *[Hint: Why does such an index i always exist?]*

7. Suppose you are given a stack of n pancakes of different sizes. You want to sort the pancakes so that smaller pancakes are on top of larger pancakes. The only operation you can perform is a *flip*—insert a spatula under the top k pancakes, for some integer k between 1 and n, and flip them all over.

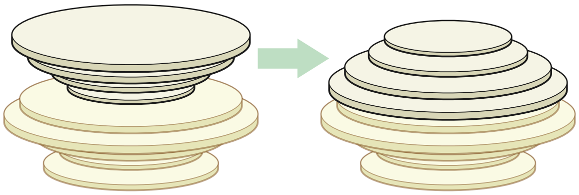


Figure 1. Flipping the top four pancakes.

(a) Describe an algorithm to sort an arbitrary stack of n pancakes using as few flips as possible. *Exactly* how many flips does your algorithm perform in the worst case?

(b) Now suppose one side of each pancake is burned. Describe an algorithm to sort an arbitrary stack of n pancakes, so that the burned side of every pancake is facing down, using as few flips as possible. *Exactly* how many flips does your algorithm perform in the worst case?

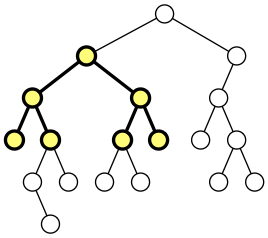
[Hint: This problem has ***nothing*** *to do with the Tower of Hanoi!]*

8. (a) Describe an algorithm to determine in $O(n)$ time whether an arbitrary array $A[1\,..\,n]$ contains more than $n/4$ copies of any value.

(b) Describe and analyze an algorithm to determine, given an arbitrary array $A[1\,..\,n]$ and an integer k, whether A contains more than k copies of any value. Express the running time of your algorithm as a function of both n and k.

Do not use hashing, or radix sort, or any other method that depends on the precise input values, as opposed to their order.

9. For this problem, a *subtree* of a binary tree means any connected subgraph. A binary tree is *complete* if every internal node has two children, and every leaf has exactly the same depth. Describe and analyze a recursive algorithm to compute the *largest complete subtree* of a given binary tree. Your algorithm should return both the root and the depth of this subtree.



The largest complete subtree of this binary tree has depth 2.

*10. Suppose you have an integer array $A[1\,..\,n]$ that *used* to be sorted, but Swedish hackers have overwritten k entries of A with random numbers. Because you carefully monitor your system for intrusions, you know *how many* entries of A are corrupted, but not *which* entries or what the values are.

Describe an algorithm to determine whether your corrupted array A contains an integer x. Your input consists of the array A, the integer k, and the target integer x. For example, if A is the following array, $k = 4$, and $x = 17$, your algorithm should return True. (The corrupted entries of the array are shaded.)

2	3	99	7	11	13	17	19	25	29	31	−5	41	43	47	53	8	61	67	71

Assume that x is not equal to any of the the corrupted values, and that all n array entries are distinct. Report the running time of your algorithm as a function of n and k. A solution only for the special case $k = 1$ is worth 5 points; a complete solution for arbitrary k is worth 10 points. *[Hint: First consider $k = 0$; then consider $k = 1$.]*

Dynamic Programming

1. ⟨⟨*Lab*⟩⟩ Describe and analyze efficient algorithms for the following problems.

   (a) Given an array $A[1..n]$ of integers, compute the length of a longest ***increasing*** subsequence of A. A sequence $B[1..\ell]$ is *increasing* if $B[i] > B[i-1]$ for every index $i \geq 2$.

   (b) Given an array $A[1..n]$ of integers, compute the length of a longest ***decreasing*** subsequence of A. A sequence $B[1..\ell]$ is *decreasing* if $B[i] < B[i-1]$ for every index $i \geq 2$.

   (c) Given an array $A[1..n]$ of integers, compute the length of a longest ***alternating*** subsequence of A. A sequence $B[1..\ell]$ is *alternating* if $B[i] < B[i-1]$ for every even index $i \geq 2$, and $B[i] > B[i-1]$ for every odd index $i \geq 3$.

   (d) Given an array $A[1..n]$ of integers, compute the length of a longest ***convex*** subsequence of A. A sequence $B[1..\ell]$ is *convex* if $B[i]-B[i-1] > B[i-1]-B[i-2]$ for every index $i \geq 3$.

   (e) Given an array $A[1..n]$, compute the length of a longest ***palindrome*** subsequence of A. Recall that a sequence $B[1..\ell]$ is a *palindrome* if $B[i] = B[\ell-i+1]$ for every index i.

2. ⟨⟨*F16, HW*⟩⟩ It's almost time to show off your flippin' sweet dancing skills! Tomorrow is the big dance contest you've been training for your entire life, except for that summer you spent with your uncle in Alaska hunting wolverines. You've obtained an advance copy of the the list of n songs that the judges will play during the contest, in chronological order.

   You know all the songs, all the judges, and your own dancing ability extremely well. For each integer k, you know that if you dance to the kth song on the schedule, you will be awarded exactly $Score[k]$ points, but then you will be physically unable to dance for the next $Wait[k]$ songs (that is, you cannot dance to songs $k+1$ through $k+Wait[k]$). The dancer with the highest total score at the end of the night wins the contest, so you want your total score to be as high as possible.

   Describe and analyze an efficient algorithm to compute the maximum total score you can achieve. The input to your sweet algorithm is the pair of arrays $Score[1..n]$ and $Wait[1..n]$.

3. ⟨⟨*S16*⟩⟩ After the Revolutionary War, Alexander Hamilton's biggest rival as a lawyer was Aaron Burr. (Sir!) In fact, the two worked next door to each other. Unlike Hamilton, Burr cannot work non-stop; every case he tries exhausts him. The bigger the case, the longer he must rest before he is well enough to take the next case. (Of course, he is willing to wait for it.) If a case arrives while Burr is resting, Hamilton snatches it up instead.

   Burr has been asked to consider a sequence of n upcoming cases. He quickly computes two arrays $profit[1..n]$ and $skip[1..n]$, where for each index i,

   - $profit[i]$ is the amount of money Burr would make by taking the ith case, and

- *skip*[i] is the number of consecutive cases Burr must skip if he accepts the ith case. That is, if Burr accepts the ith case, he cannot accept cases $i+1$ through $i+skip[i]$.

Design and analyze an algorithm that determines the maximum total profit Burr can secure from these n cases, using his two arrays as input.

4. ⟨⟨*S14*⟩⟩ Recall that a *palindrome* is any string that is the same as its reversal. For example, I, DAD, HANNAH, AIBOHPHOBIA (fear of palindromes), and the empty string are all palindromes.

   (a) Describe and analyze an algorithm to find the length of the longest substring (not *subsequence*!) of a given input string that is a palindrome. For example, $\mathtt{\underline{\mathbf{BASEESAB}}}$ is the longest palindrome substring of $\mathtt{BUB\underline{\mathbf{BASEESAB}}ANANA}$ ("Bubba sees a banana."). Thus, given the input string BUBBASEESABANANA, your algorithm should return the integer 8.

   (b) ⟨⟨*Lab, F16*⟩⟩ Describe and analyze an algorithm to find the length of the longest subsequence (not *substring*!) of a given input string that is a palindrome. For example, the longest palindrome subsequence of $\mathtt{\underline{\mathbf{M}}A\underline{\mathbf{H}}D\underline{\mathbf{Y}}NA\underline{\mathbf{M}}ICP\underline{\mathbf{RO}}G\underline{\mathbf{R}}AMZLET\underline{\mathbf{M}}ESHOW\underline{\mathbf{Y}}OUT\underline{\mathbf{H}}E\underline{\mathbf{M}}}$ is MHYMRORMYHM, so given that string as input, your algorithm should output the number 11.

   (c) ⟨⟨*HW*⟩⟩ Any string can be decomposed into a sequence of palindrome substrings. For example, the string BUBBASEESABANANA can be broken into palindromes in the following ways (and many others):

   $$\mathtt{BUB} + \mathtt{BASEESAB} + \mathtt{ANANA}$$
   $$\mathtt{B} + \mathtt{U} + \mathtt{BB} + \mathtt{A} + \mathtt{SEES} + \mathtt{ABA} + \mathtt{NAN} + \mathtt{A}$$
   $$\mathtt{B} + \mathtt{U} + \mathtt{BB} + \mathtt{A} + \mathtt{SEES} + \mathtt{A} + \mathtt{B} + \mathtt{ANANA}$$
   $$\mathtt{B} + \mathtt{U} + \mathtt{B} + \mathtt{B} + \mathtt{A} + \mathtt{S} + \mathtt{E} + \mathtt{E} + \mathtt{S} + \mathtt{A} + \mathtt{B} + \mathtt{A} + \mathtt{N} + \mathtt{A} + \mathtt{N} + \mathtt{A}$$

   Describe and analyze an algorithm to find the smallest number of palindromes that make up a given input string. For example, if your input is the string BUBBASEESABANANA, your algorithm should return the integer 3.

5. ⟨⟨*F16*⟩⟩ A *shuffle* of two strings X and Y is formed by interspersing the characters into a new string, keeping the characters of X and Y in the same order. For example, the string BANANAANANAS is a shuffle of the strings BANANA and ANANAS in several different ways.

   $$\mathtt{BANANA}_{\mathtt{ANANAS}} \qquad \mathtt{BAN}_{\mathtt{ANA}}{}^{\mathtt{ANA}}{}_{\mathtt{NAS}} \qquad \mathtt{B}_{\mathtt{AN}}{}^{\mathtt{AN}}{}_{\mathtt{A}}{}^{\mathtt{A}}{}_{\mathtt{NA}}{}^{\mathtt{NA}}{}_{\mathtt{S}}$$

   Similarly, the strings PRODGYRNAMAMMIINCG and DYPRONGARMAMMICING are both shuffles of DYNAMIC and PROGRAMMING:

   $$\mathtt{PRO}^{\mathtt{D}}\mathtt{G}^{\mathtt{Y}}\mathtt{R}^{\mathtt{NAM}}\mathtt{AMMI}^{\mathtt{I}}\mathtt{N}^{\mathtt{C}}\mathtt{G} \qquad {}^{\mathtt{DY}}\mathtt{PRO}^{\mathtt{N}}\mathtt{G}^{\mathtt{A}}\mathtt{R}^{\mathtt{M}}\mathtt{AMM}^{\mathtt{IC}}\mathtt{ING}$$

   Describe and analyze an efficient algorithm to determine, given three strings $A[1..m]$, $B[1..n]$, and $C[1..m+n]$, whether C is a shuffle of A and B.

6. Suppose we are given an n-digit integer X. Repeatedly remove one digit from either end of X (your choice) until no digits are left. The *square-depth* of X is the maximum number of perfect squares that you can see during this process. For example, the number 32492 has square-depth 3, by the following sequence of removals:

$$32492 \to \underline{\mathbf{3249}}\cancel{2} \to \underline{\mathbf{324}}\cancel{9} \to \cancel{3}24 \to \cancel{2}\underline{\mathbf{4}} \to \cancel{4}.$$

Describe and analyze an algorithm to compute the square-depth of a given integer X, represented as an array $X[1..n]$ of n decimal digits. Assume you have access to a subroutine IsSquare that determines whether a given k-digit number (represented by an array of digits) is a perfect square ***in*** $\boldsymbol{O(k^2)}$ ***time***.

7. Suppose you are given a sequence of non-negative integers separated by + and × signs; for example:

$$2 \times 3 + 0 \times 6 \times 1 + 4 \times 2$$

You can change the value of this expression by adding parentheses in different places. For example:

$$2 \times (3 + (0 \times (6 \times (1 + (4 \times 2))))) = 6$$
$$(((((2 \times 3) + 0) \times 6) \times 1) + 4) \times 2 = 80$$
$$((2 \times 3) + (0 \times 6)) \times (1 + (4 \times 2)) = 108$$
$$(((2 \times 3) + 0) \times 6) \times ((1 + 4) \times 2) = 360$$

Describe and analyze an algorithm to compute, given a list of integers separated by + and × signs, the smallest possible value we can obtain by inserting parentheses.

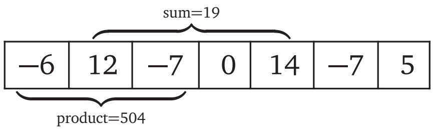
Your input is an array $A[0..2n]$ where each $A[i]$ is an integer if i is even and + or × if i is odd. Assume any arithmetic operation in your algorithm takes $O(1)$ time.

8. Suppose you are given an array $A[1..n]$ of numbers, which may be positive, negative, or zero, and which are ***not*** necessarily integers.

   (a) Describe and analyze an algorithm that finds the largest sum of of elements in a contiguous subarray $A[i..j]$.

   (b) Describe and analyze an algorithm that finds the largest *product* of of elements in a contiguous subarray $A[i..j]$.

For example, given the array $[-6, 12, -7, 0, 14, -7, 5]$ as input, your first algorithm should return the integer 19, and your second algorithm should return the integer 504.



For the sake of analysis, assume that comparing, adding, or multiplying any pair of numbers takes $O(1)$ time.

[Hint: Problem (a) has been a standard computer science interview question since at least the mid-1980s. You can find many correct solutions on the web; the problem even has its own Wikipedia page! But at least in 2016, a significant fraction of the solutions I found on the web for problem (b) were either significantly slower than necessary or actually incorrect. Remember that the product of two negative numbers is positive.]

9. Suppose you are given three strings $A[1\,..\,n]$, $B[1\,..\,n]$, and $C[1\,..\,n]$.

   (a) Describe and analyze an algorithm to find the length of the longest common subsequence of all three strings. For example, given the input strings

   $$A = \texttt{AxxBxxCDxEF}, \qquad B = \texttt{yyABCDyEyFy}, \qquad C = \texttt{zAzzBCDzEFz},$$

   your algorithm should output the number 6, which is the length of the longest common subsequence `ABCDEF`.

   (b) Describe and analyze an algorithm to find the length of the shortest common supersequence of all three strings. For example, given the input strings

   $$A = \texttt{AxxBxxCDxEF}, \qquad B = \texttt{yyABCDyEyFy}, \qquad C = \texttt{zAzzBCDzEFz},$$

   your algorithm should output the number 21, which is the length of the shortest common supersequence `yzyAxzzxBxxCDxyzEyFzy`.

10. ⟨⟨*S18*⟩⟩ Suppose we want to split an array $A[1\,..\,n]$ of integers into k contiguous intervals that partition the sum of the values as evenly as possible. Specifically, define the *cost* of such a partition as the maximum, over all k intervals, of the sum of the values in that interval; our goal is to minimize this cost. Describe and analyze an algorithm to compute the minimum cost of a partition of A into k intervals, given the array A and the integer k as input.

    For example, given the array $A = [1, 6, -1, 8, 0, 3, 3, 9, 8, 8, 7, 4, 9, 8, 9, 4, 8, 4, 8, 2]$ and the integer $k = 3$ as input, your algorithm should return the integer 37, which is the cost of the following partition:

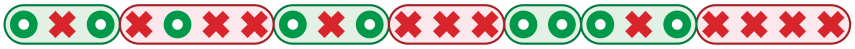
    $$\Big[\, \overbrace{1, 6, -1, 8, 0, 3, 3, 9, 8}^{37} \;\Big|\; \overbrace{8, 7, 4, 9, 8}^{36} \;\Big|\; \overbrace{9, 4, 8, 4, 8, 2}^{35} \,\Big]$$

    The numbers above each interval show the sum of the values in that interval.

11. ⟨⟨*S18*⟩⟩ The City Council of Sham-Poobanana needs to partition Purple Street into voting districts. A total of n people live on Purple Street, at consecutive addresses $1, 2, \ldots, n$. Each voting district must be a contiguous interval of addresses $i, i+1, \ldots, j$ for some $1 \le i < j \le n$. By law, each Purple Street address must lie in exactly one district, and the number of addresses in each district must be between k and $2k$, where k is some positive integer parameter.

    Every election in Sham-Poobanana is between two rival factions: Oceania and Eurasia. A majority of the City Council are from Oceania, so they consider a district to be *good* if more than half the residents of that district voted for Oceania in the previous election. Naturally, the City Council has complete voting records for all n residents.

For example, the figure below shows a legal partition of 22 addresses into 4 good districts and 3 bad districts, where $k = 2$. Each O indicates a vote for Oceania, and each X indicates a vote for Eurasia.



Describe an algorithm to find the largest possible number of *good* districts in a legal partition. Your input consists of the integer k and a boolean array $GoodVote[1..n]$ indicating which residents previously voted for Oceania (TRUE) or Eurasia (FALSE). You can assume that a legal partition exists. Analyze the running time of your algorithm in terms of the parameters n and k.

12. (a) Suppose we are given a set L of n line segments in the plane, where each segment has one endpoint on the line $y = 0$ and one endpoint on the line $y = 1$, and all $2n$ endpoints are distinct. Describe and analyze an algorithm to compute the largest subset of L in which no pair of segments intersects.

    (b) Suppose we are given a set L of n line segments in the plane, where each segment has one endpoint on the line $y = 0$ and one endpoint on the line $y = 1$, and all $2n$ endpoints are distinct. Describe and analyze an algorithm to compute the largest subset of L in which ***every*** pair of segments intersects.

13. Suppose you are given an $m \times n$ bitmap, represented by an array $M[1..n, 1..n]$ of 0s and 1s. A *solid square block* in M is a subarray of the form $M[i..i+w, j..j+w]$ containing only 1-bits. Describe and analyze an algorithm to find the largest solid square block in M.

14. You and your six-year-old nephew Elmo decide to play a simple card game. At the beginning of the game, the cards are dealt face up in a long row. Each card is worth a different number of points. After all the cards are dealt, you and Elmo take turns removing either the leftmost or rightmost card from the row, until all the cards are gone. At each turn, you can decide which of the two cards to take. The winner of the game is the player that has collected the most points when the game ends.

    Having never taken an algorithms class, Elmo follows the obvious greedy strategy—when it's his turn, Elmo *always* takes the card with the higher point value. Your task is to find a strategy that will beat Elmo whenever possible. (It might seem mean to beat up on a little kid like this, but Elmo absolutely *hates* it when grown-ups let him win.)

    (a) Prove that you should not also use the greedy strategy. That is, show that there is a game that you can win, but only if you do *not* follow the same greedy strategy as Elmo.

    (b) Describe and analyze an algorithm to determine, given the initial sequence of cards, the maximum number of points that you can collect playing against Elmo.

(c) Five years later, thirteen-year-old Elmo has become a *much* stronger player. Describe and analyze an algorithm to determine, given the initial sequence of cards, the maximum number of points that you can collect playing against a *perfect* opponent.

15. ⟨⟨***S16***⟩⟩ Your nephew Elmo is visiting you for Christmas, and he's brought a different card game. Like your previous game with Elmo, this game is played with a row of n cards, each labeled with an integer (which could be positive, negative, or zero). Both players can see all n card values. Otherwise, the game is almost completely different.

On each turn, the current player must take the leftmost card. The player can either keep the card or give it to their opponent. If they keep the card, their turn ends and their opponent takes the next card; however, if they give the card to their opponent, the current player's turn continues with the next card. In short, the player that does *not* get the ith card decides who gets the $(i+1)$th card. The game ends when all cards have been played. Each player adds up their card values, and whoever has the higher total wins.

For example, suppose the initial cards are $[3, -1, 4, 1, 5, 9]$, and Elmo plays first. Then the game might proceed as follows:

- Elmo keeps the 3, ending his turn.
- You give Elmo the -1.
- You keep the 4, ending your turn.
- Elmo gives you the 1.
- Elmo gives you the 5.
- Elmo keeps the 9, ending his turn. All cards are gone, so the game is over.
- Your score is $1+4+5=10$ and Elmo's score is $3-1+9=11$, so Elmo wins.

Describe an algorithm to compute the highest possible score you can earn from a given row of cards, assuming Elmo plays first and plays perfectly. Your input is the array $C[1..n]$ of card values. For example, if the input is $[3, -1, 4, 1, 5, 9]$, your algorithm should return the integer 10.

16. ⟨⟨***F14***⟩⟩ The new swap-puzzle game *Candy Swap Saga XIII* involves n cute animals numbered 1 through n. Each animal holds one of three types of candy: circus peanuts, Heath bars, and Cioccolateria Gardini chocolate truffles. You also have a candy in your hand; at the start of the game, you have a circus peanut.

To earn points, you visit each of the animals in order from 1 to n. For each animal, you can either keep the candy in your hand or exchange it with the candy the animal is holding.

- If you swap your candy for another candy of the *same* type, you earn one point.
- If you swap your candy for a candy of a *different* type, you lose one point. (Yes, your score can be negative.)
- If you visit an animal and decide not to swap candy, your score does not change.

You *must* visit the animals in order, and once you visit an animal, you can never visit it again.

Describe and analyze an efficient algorithm to compute your maximum possible score. Your input is an array $C[1..n]$, where $C[i]$ is the type of candy that the ith animal is holding.

17. ⟨⟨*F14, HW*⟩⟩ Farmers Boggis, Bunce, and Bean have set up an obstacle course for Mr. Fox. The course consists of a row of n booths, each with an integer painted on the front with bright red paint, which could be positive, negative, or zero. Let $A[i]$ denote the number painted on the front of the ith booth. Everyone has agreed to the following rules:

    - At each booth, Mr. Fox ***must*** say either "Ring!" or "Ding!".
    - If Mr. Fox says "Ring!" at the ith booth, he earns a reward of $A[i]$ chickens. (If $A[i] < 0$, Mr. Fox pays a penalty of $-A[i]$ chickens.)
    - If Mr. Fox says "Ding!" at the ith booth, he pays a penalty of $A[i]$ chickens. (If $A[i] < 0$, Mr. Fox earns a reward of $-A[i]$ chickens.)
    - Mr. Fox is forbidden to say the same word more than three times in a row. For example, if he says "Ring!" at booths 6, 7, and 8, then he *must* say "Ding!" at booth 9.
    - All accounts will be settled at the end; Mr. Fox does not actually have to carry chickens through the obstacle course.
    - If Mr. Fox violates any of the rules, or if he ends the obstacle course owing the farmers chickens, the farmers will shoot him.

    Describe and analyze an algorithm to compute the largest number of chickens that Mr. Fox can earn by running the obstacle course, given the array $A[1..n]$ of booth numbers as input.

**Greedy algorithms will not be covered on this semester's exams.
These problems are provided only for reference.**

Greedy Algorithms

Remember that you will receive ***zero*** points for a greedy algorithm, even if it is perfectly correct, unless you also give a formal proof of correctness.

1. ⟨⟨*Lab*⟩⟩ Recall the class scheduling problem described in lecture on Tuesday. We are given two arrays $S[1..n]$ and $F[1..n]$, where $S[i] < F[i]$ for each i, representing the start and finish times of n classes. Your goal is to find the largest number of classes you can take without ever taking two classes simultaneously. We showed in class that the following greedy algorithm constructs an optimal schedule:

   Choose the course that *ends first*, discard all conflicting classes, and recurse.

   But this is not the only greedy strategy we could have tried. For each of the following alternative greedy algorithms, either prove that the algorithm always constructs an optimal schedule, or describe a small input example for which the algorithm does not produce an optimal schedule. Assume that all algorithms break ties arbitrarily (that is, in a manner that is completely out of your control).

   [Hint: Exactly three of these greedy strategies actually work.]

   (a) Choose the course x that *ends last*, discard classes that conflict with x, and recurse.

   (b) Choose the course x that *starts first*, discard all classes that conflict with x, and recurse.

   (c) Choose the course x that *starts last*, discard all classes that conflict with x, and recurse.

   (d) Choose the course x with *shortest duration*, discard all classes that conflict with x, and recurse.

   (e) Choose a course x that *conflicts with the fewest other courses*, discard all classes that conflict with x, and recurse.

   (f) If no classes conflict, choose them all. Otherwise, discard the course with *longest duration* and recurse.

   (g) If no classes conflict, choose them all. Otherwise, discard a course that *conflicts with the most other courses* and recurse.

   (h) Let x be the class with the *earliest start time*, and let y be the class with the *second earliest start time*.

   - If x and y are disjoint, choose x and recurse on everything but x.
   - If x completely contains y, discard x and recurse.
   - Otherwise, discard y and recurse.

   (i) If any course x completely contains another course, discard x and recurse. Otherwise, choose the course y that *ends last*, discard all classes that conflict with y, and recurse.

2. ⟨⟨*S14*⟩⟩ Binaria uses coins whose values are $1, 2, 4, \ldots, 2^k$, the first k powers of two, for some integer k. As in most countries, Binarian shopkeepers always make change using the following greedy algorithm:

```
MakeChange(N):
  if N = 0
      say "Thank you, come again!"
  else
      c ← largest coin value such that c ≤ N
      give the customer one c cent coin
      MakeChange(N − c)
```

For example, to make 37 cents in change, the shopkeeper would give the customer one 32 cent coin, one 4 cent coin, and one 1 cent coin, and then say "Thank you, come again!" (For purposes of this problem, assume that every shopkeeper has an unlimited supply of each type of coin.)

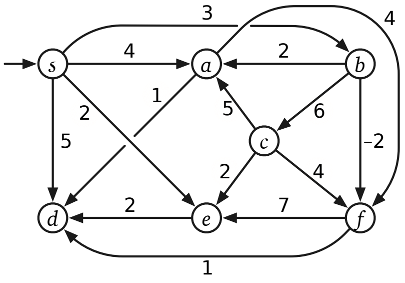
Prove that this greedy algorithm always uses the smallest possible number of coins. *[Hint: Prove that the greedy algorithm uses at most one coin of each denomination.]*

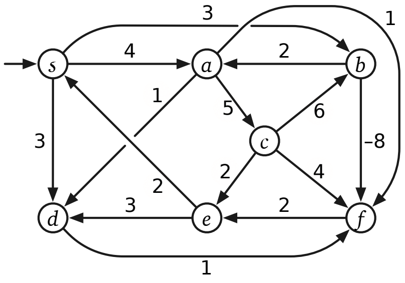
3. Let X be a set of n intervals on the real line. We say that a set P of points *stabs* X if every interval in X contains at least one point in P. Describe and analyze an efficient algorithm to compute the smallest set of points that stabs X. Assume that your input consists of two arrays $L[1..n]$ and $R[1..n]$, representing the left and right endpoints of the intervals in X. If you use a greedy algorithm, don't forget to ***prove*** that it is correct.

Graph Algorithms

Sanity Check

1. ⟨⟨***S14, F14, F16***⟩⟩ Indicate the following structures in the example graphs below. If the requested structure does not exist, just write None. To indicate a subgraph, draw over the entire edge with a heavy black line; your answer should be visible from across the room.





(a) A depth-first spanning tree rooted at node s.

(b) A breadth-first spanning tree rooted at node s.

(c) A shortest-path tree rooted at node s.

(d) The set of all vertices reachable from node c. (Circle each vertex.)

(e) The set of all vertices that can reach node c. (Circle each vertex.)

(f) The strongly connected components. (Circle each one.)

(g) A simple cycle containing vertex s.

(h) The shortest directed cycle.

(i) A depth-first pre-ordering of the vertices. (List the vertices in order.)

(j) A depth-first post-ordering of the vertices. (List the vertices in order.)

(k) A topological order. (List the vertices in order.)

(l) A walk from s to d with the maximum number of edges.

(m) A walk from s to d with the largest total weight.

[On an actual exam, we would only ask about one graph, we would not ask for all these structures, and we would give you several copies of the graph on which to mark your answers.]

Reachability/Connectivity/Traversal

1. Describe and analyze algorithms for the following problems; in each problem, you are given a graph $G = (V, E)$ with unweighted edges, which may be directed or undirected. You may or may not need different algorithms for directed and undirected graphs.

   (a) Find two vertices that are (strongly) connected.

   (b) Find two vertices that are not (strongly) connected.

   (c) Find two vertices, such that neither can reach the other.

   (d) Find all vertices reachable from a given vertex s.

   (e) Find all vertices that can reach a given vertex s.

   (f) Find all vertices that are strongly connected to a given vertex s.

   (g) Find a simple cycle, or correctly report that the graph has no cycles. (A simple cycle is a closed walk that visits each vertex at most once.)

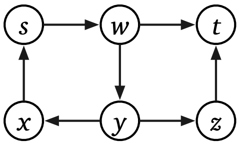
   (h) Find the *shortest* simple cycle, or correctly report that the graph has no cycles.

   (i) Determine whether deleting a given vertex v would disconnect the graph.

   [On an actual exam, we would not ask for all these structures, and we would specify whether the input graph is directed or undirected.]

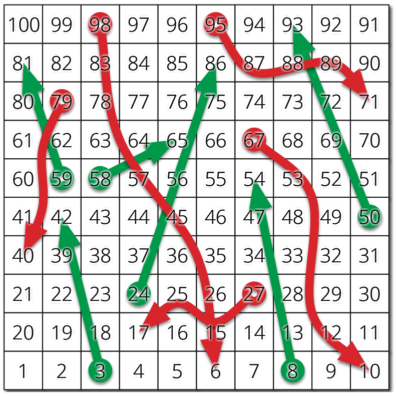
2. ⟨⟨*F14*⟩⟩ Suppose you are given a directed graph $G = (V, E)$ and two vertices s and t. Describe and analyze an algorithm to determine if there is a walk in G from s to t (possibly repeating vertices and/or edges) whose length is divisible by 3.

   For example, given the graph below, with the indicated vertices s and t, your algorithm should return True, because the walk $s \to w \to y \to x \to s \to w \to t$ has length 6.



   [Hint: Build a (different) graph.]

3. ⟨⟨*Lab*⟩⟩ ***Snakes and Ladders*** is a classic board game, originating in India no later than the 16th century. The board consists of an $n \times n$ grid of squares, numbered consecutively from 1 to n^2, starting in the bottom left corner and proceeding row by row from bottom to top, with rows alternating to the left and right. Certain pairs of squares, always in different rows, are connected by either "snakes" (leading down) or "ladders" (leading up). Each square can be an endpoint of at most one snake or ladder.



A typical Snakes and Ladders board.
Upward straight arrows are ladders; downward wavy arrows are snakes.

You start with a token in cell 1, in the bottom left corner. In each move, you advance your token up to k positions, for some fixed constant k (typically 6). If the token ends the move at the *top* end of a snake, you ***must*** slide the token down to the bottom of that snake. If the token ends the move at the *bottom* end of a ladder, you ***may*** move the token up to the top of that ladder.

Describe and analyze an algorithm to compute the smallest number of moves required for the token to reach the last square of the grid.

4. Let G be a connected undirected graph. Suppose we start with two coins on two arbitrarily chosen vertices of G, and we want to move the coins so that they lie on the same vertex using as few moves as possible. At every step, each coin *must* move to an adjacent vertex.

   (a) ⟨⟨***Lab***⟩⟩ Describe and analyze an algorithm to compute the minimum number of steps to reach a configuration where both coins are on the same vertex, or to report correctly that no such configuration is reachable. The input to your algorithm consists of a graph $G = (V, E)$ and two vertices $u, v \in V$ (which may or may not be distinct).

   (b) Now suppose there are three coins. Describe and analyze an algorithm to compute the minimum number of steps to reach a configuration where both coins are on the same vertex, or to report correctly that no such configuration is reachable. *[Hint: Some people already considered this problem in lab.]*

   (c) ⟨⟨***Lab***⟩⟩ Finally, suppose there are *forty-two* coins. Describe and analyze an algorithm to determine whether it is possible to move all 42 coins to the same vertex. Again, *every* coin must move at *every* step. For full credit, your algorithm should run in $O(V + E)$ time.

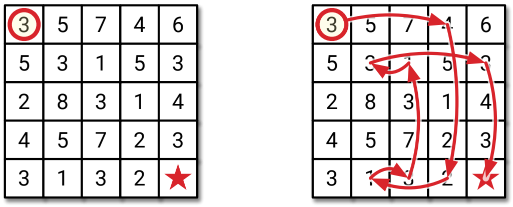
5. A graph (V, E) is bipartite if the vertices V can be partitioned into two subsets L and R, such that every edge has one vertex in L and the other in R.

   (a) Prove that every tree is a bipartite graph.

   (b) Describe and analyze an efficient algorithm that determines whether a given undirected graph is bipartite.

6. ⟨⟨*F14, S18*⟩⟩ A ***number maze*** is an $n \times n$ grid of positive integers. A token starts in the upper left corner; your goal is to move the token to the lower-right corner. On each turn, you are allowed to move the token up, down, left, or right; the distance you may move the token is determined by the number on its current square. For example, if the token is on a square labeled 3, then you may move the token three steps up, three steps down, three steps left, or three steps right. However, you are never allowed to move the token off the edge of the board.

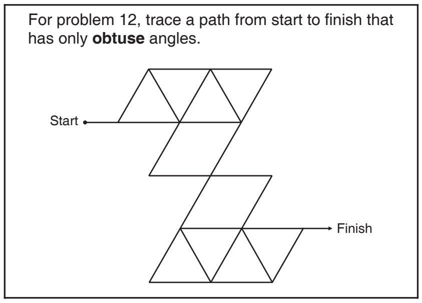
   Describe and analyze an efficient algorithm that either returns the minimum number of moves required to solve a given number maze, or correctly reports that the maze has no solution. For example, given the maze shown below, your algorithm would return the number 8.



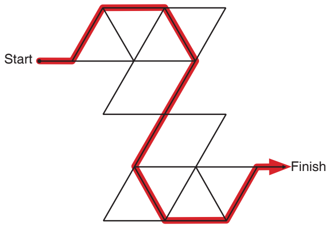
3	5	7	4	6
5	3	1	5	3
2	8	3	1	4
4	5	7	2	3
3	1	3	2	★

A 5 × 5 number maze that can be solved in eight moves.

7. ⟨⟨*F16*⟩⟩ The following puzzle appears in my younger daughter's math workbook.[1] (I've put the solution on the right so you don't waste time solving it during the exam.)



For problem 12, trace a path from start to finish that has only **obtuse** angles.



Describe and analyze an algorithm to solve arbitrary obtuse-angle mazes.

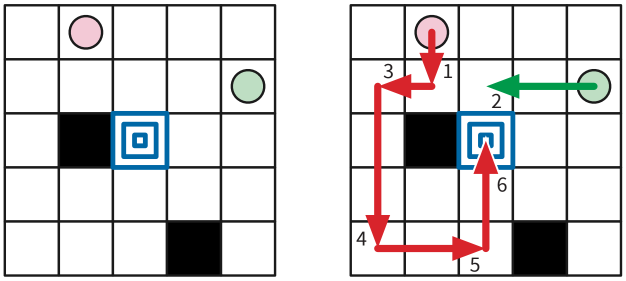
You are given a connected undirected graph G, whose vertices are points in the plane and whose edges are line segments. Edges do not intersect, except at their endpoints. For example, a drawing of the letter X would have five vertices and four edges; the maze above has 17 vertices and 26 edges. You are also given two vertices Start and Finish.

Your algorithm should return True if G contains a walk from Start to Finish that has only obtuse angles, and False otherwise. Formally, a walk through G is valid if $\pi/2 < \angle uvw \leq \pi$ for every pair of consecutive edges $u{\rightarrow}v{\rightarrow}w$ in the walk. Assume you have a subroutine that can determine whether the angle between any two segments is acute, right, obtuse, or straight in $O(1)$ time.

[1] Jason Batterson and Shannon Rogers, *Beast Academy Math: Practice 3A*, 2012. See https://www.beastacademy.com/resources/printables.php for more examples.

8. ***Kaniel Dane*** is a solitaire puzzle played with two tokens on an $n \times n$ square grid. Some squares of the grid are marked as *obstacles*, and one grid square is marked as the *target*. In each turn, the player must move one of the tokens from is current position *as far as possible* upward, downward, right, or left, stopping just before the token hits (1) the edge of the board, (2) an obstacle square, or (3) the other token. The goal is to move either of the tokens onto the target square.

   For example, in the instance below, we move the red token down until it hits the obstacle, then move the green token left until it hits the red token, and then move the red token left, down, right, and up. In the last move, the red token stops at the target *because* the green token is on the next square above.



An instance of the Kaniel Dane puzzle that can be solved in six moves.
Circles indicate the initial token positions; black squares are obstacles; the center square is the target.

   Describe and analyze an algorithm to determine whether an instance of this puzzle is solvable. Your input consist of the integer n, a list of obstacle locations, the target location, and the initial locations of the tokens. The output of your algorithm is a single boolean: TRUE if the given puzzle is solvable and FALSE otherwise. The running time of your algorithm should be a small polynomial in n.

9. ⟨⟨*F16*⟩⟩ Suppose you have a collection of n lockboxes and m gold keys. Each key unlocks *at most* one box; however, each box might be unlocked by one key, by multiple keys, or by no keys at all. There are only two ways to open each box once it is locked: Unlock it properly (which requires having a matching key in your hand), or smash it to bits with a hammer.

   Your baby brother, who loves playing with shiny objects, has somehow managed to lock all your keys inside the boxes! Luckily, your home security system recorded everything, so you know exactly which keys (if any) are inside each box. You need to get all the keys back out of the boxes, because they are made of gold. Clearly you have to smash at least one box.

   (a) Your baby brother has found the hammer and is eagerly eyeing one of the boxes. Describe and analyze an algorithm to determine if smashing the box your brother has chosen would allow you to retrieve all m keys.

   (b) Describe and analyze an algorithm to compute the minimum number of boxes that must be smashed to retrieve all the keys. *[Hint: This subproblem should really be in the next section.]*

Depth-First Search, Dags, Strong Connectivity

1. ⟨⟨*Lab*⟩⟩ Inspired by an earlier question, you decided to organize a Snakes and Ladders competition with n participants. In this competition, each game of Snakes and Ladders involves three players. After the game is finished, they are ranked first, second and third. Each player may be involved in any (non-negative) number of games, and the number needs not be equal among players.

   At the end of the competition, m games have been played. You realized that you had forgotten to implement a proper rating system, and therefore decided to produce the overall ranking of all n players as you see fit. However, to avoid being too suspicious, if player A ranked better than player B in any game, then A must rank better than B in the overall ranking.

   You are given the list of players involved and the ranking in each of the m games. Describe and analyze an algorithm to produce an overall ranking of the n players that satisfies the condition, or correctly reports that it is impossible.

2. Let G be a directed acyclic graph with a unique source s and a unique sink t.

   (a) A *Hamiltonian path* in G is a directed path in G that contains every vertex in G. Describe an algorithm to determine whether G has a Hamiltonian path.

   (b) Suppose the vertices of G have weights. Describe an efficient algorithm to find the path from s to t with maximum total weight.

   (c) Suppose we are also given an integer ℓ. Describe an efficient algorithm to find the maximum-weight path from s to t, such that the path contains at most ℓ edges. (Assume there is at least one such path.)

   (d) Suppose several vertices in G are marked *essential*, and we are given an integer k. Design an efficient algorithm to determine whether there is a path from s to t that passes through at least k essential vertices.

   (e) Suppose the vertices of G have integer labels, where $label(s) = -\infty$ and $label(t) = \infty$. Describe an algorithm to find the path from s to t with the maximum number of edges, such that the vertex labels define an increasing sequence.

   (f) ⟨⟨*Lab*⟩⟩ Describe an algorithm to compute the number of distinct paths from s to t in G. (Assume that you can add arbitrarily large integers in $O(1)$ time.)

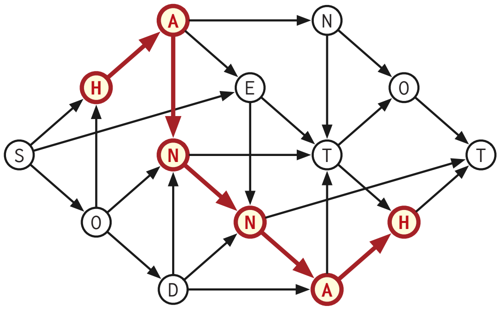
3. Suppose you are given a directed graph G in which ***every edge has negative weight***, and a source vertex s. Describe and analyze an efficient algorithm that computes the shortest path distances from s to every other vertex in G. Specifically, for every vertex t:

   - If t is not reachable from s, your algorithm should report $dist(t) = \infty$.
   - If the shortest-path distance from s to t is not well-defined because of negative cycles, your algorithm should report $dist(t) = -\infty$.
   - If neither of the two previous conditions applies, your algorithm should report the correct shortest-path distance from s to t.

[Hint: First think about graphs where the first two conditions never happen.]

4. Let G be a directed acyclic graph whose vertices have labels from some fixed alphabet. Any directed path in G has a label, which is a string obtained by concatenating the labels of its vertices. Recall that a *palindrome* is a string that is equal to its reversal.

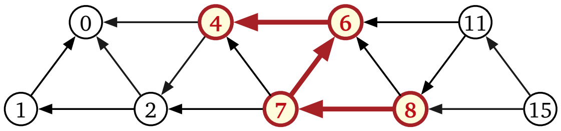
   Describe and analyze an algorithm to find the length of the longest palindrome that is the label of a path in G. For example, given the dag below, your algorithm should return the integer 6, which is the length of the palindrome HANNAH.



5. ⟨⟨*S18*⟩⟩ Let G be a ***directed*** graph, where every vertex v has an associated height $h(v)$, and for every edge $u \to v$ we have the inequality $h(u) > h(v)$. Assume all heights are distinct. The *span* of a path from u to v is the height difference $h(u) - h(v)$.

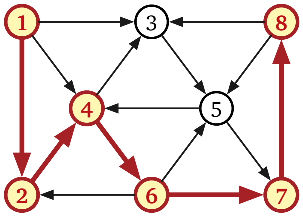
   Describe and analyze an algorithm to find the ***minimum span*** of a path in G with ***at least*** k edges. Your input consists of the graph G, the vertex heights $h(\cdot)$, and the integer k. Report the running time of your algorithm as a function of V, E, and k.

   For example, given the following labeled graph and the integer $k = 3$ as input, your algorithm should return the integer 4, which is the span of the path 8→7→6→4.



6. ⟨⟨*S18*⟩⟩ Let G be an arbitrary (*not* necessarily acyclic) directed graph in which every vertex v has an integer label $\ell(v)$. Describe an algorithm to find the longest directed path in G whose vertex labels define an increasing sequence. Assume all labels are distinct.

   For example, given the following graph as input, your algorithm should return the integer 5, which is the length of the increasing path 1→2→4→6→7→8.



Shortest Paths

1. ⟨⟨*F14*⟩⟩ Let G be a directed graph with weighted edges, and let s be a vertex of G. Suppose every vertex $v \neq s$ stores a pointer $pred(v)$ to another vertex in G. Describe and analyze an algorithm to determine whether these predecessor pointers correctly define a single-source shortest path tree rooted at s. Do ***not*** assume that G has no negative cycles.

2. ⟨⟨*F14*⟩⟩ Suppose we are given an undirected graph G in which every *vertex* has a positive weight.

   (a) Describe and analyze an algorithm to find a *spanning tree* of G with minimum total weight. (The total weight of a spanning tree is the sum of the weights of its vertices.)

   (b) Describe and analyze an algorithm to find a *path* in G from one given vertex s to another given vertex t with minimum total weight. (The total weight of a path is the sum of the weights of its vertices.)

3. ⟨⟨*S14, S18, Lab*⟩⟩ You just discovered your best friend from elementary school on Twitbook. You both want to meet as soon as possible, but you live in two different cites that are far apart. To minimize travel time, you agree to meet at an intermediate city, and then you simultaneously hop in your cars and start driving toward each other. But where *exactly* should you meet?

   You are given a weighted graph $G = (V, E)$, where the vertices V represent cities and the edges E represent roads that directly connect cities. Each edge e has a weight $w(e)$ equal to the time required to travel between the two cities. You are also given a vertex p, representing your starting location, and a vertex q, representing your friend's starting location.

   Describe and analyze an algorithm to find the target vertex t that allows you and your friend to meet as quickly as possible.

4. ⟨⟨*F16*⟩⟩ There are n galaxies connected by m intergalactic teleport-ways. Each teleport-way joins two galaxies and can be traversed in both directions. Also, each teleport-way uv has an associated cost of $c(uv)$ galactic credits, for some positive integer $c(uv)$. The same teleport-way can be used multiple times in either direction, but the same toll must be paid every time it is used.

   Judy wants to travel from galaxy s to galaxy t, but teleportation is rather unpleasant, so she wants to minimize the number of times she has to teleport. However, she also wants the total cost to be a multiple of 10 galactic credits, because carrying small change is annoying.

   Describe and analyze an algorithm to compute the minimum number of times Judy must teleport to travel from galaxy s to galaxy t so that the total cost of all teleports is an integer multiple of 10 galactic credits. Your input is a graph $G = (V, E)$ whose vertices are galaxies and whose edges are teleport-ways; every edge uv in G stores the corresponding cost $c(uv)$.

   *[Hint: This is **not** the same Intergalactic Judy problem that you saw in lab.]*

5. ⟨⟨*Lab*⟩⟩ A *looped tree* is a weighted, directed graph built from a binary tree by adding an edge from every leaf back to the root. Every edge has non-negative weight.

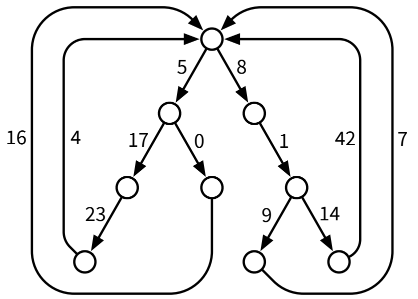


Figure 2. A looped tree.

   (a) How much time would Dijkstra's algorithm require to compute the shortest path between two vertices u and v in a looped tree with n nodes?

   (b) Describe and analyze a faster algorithm.

6. ⟨⟨*F17, Lab*⟩⟩ Suppose you are given a directed graph G with weighted edges, where *exactly one* edge has negative weight and all other edge weights are positive, along with two vertices s and t. Describe and analyze an algorithm that either computes a shortest path in G from s to t, or reports correctly that the G contains a negative cycle. (As always, faster algorithms are worth more points.)

7. After graduating you accept a job with Aerophobes-Я-Us, the leading traveling agency for people who hate to fly. Your job is to build a system to help customers plan airplane trips from one city to another. All of your customers are afraid of flying (and by extension, airports), so any trip you plan needs to be as short as possible. You know all the departure and arrival times of all the flights on the planet.

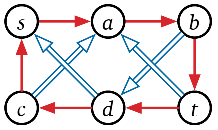
   Suppose one of your customers wants to fly from city X to city Y. Describe an algorithm to find a sequence of flights that minimizes the *total time in transit*—the length of time from the initial departure to the final arrival, including time at intermediate airports waiting for connecting flights.

8. When there is more than one shortest path from one node s to another node t, it is often convenient to choose a shortest path with the fewest edges; call this the ***best*** path from s to t. Suppose we are given a directed graph G with positive edge weights and a source vertex s in G. Describe and analyze an algorithm to compute best paths in G from s to every other vertex.

9. ⟨⟨*S18*⟩⟩ Suppose you are given a directed graph G where some edges are red and the remaining edges are blue. Describe an algorithm to find the shortest walk in G from one vertex s to another vertex t in which no three consecutive edges have the same color. That

is, if the walk contains two red edges in a row, the next edge must be blue, and if the walk contains two blue edges in a row, the next edge must be red.

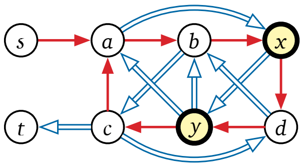
For example, if you are given the graph below (where single arrows are red and double arrows are blue), your algorithm should return the integer 7, because the shortest legal walk from s to t is $s{\to}a{\to}b{\Rightarrow}d{\to}c{\Rightarrow}a{\to}b{\to}c$.



10. ⟨⟨***S18***⟩⟩ Suppose you are given a directed graph G in which every edge is either red or blue, and a subset of the vertices are marked as *special*. A walk in G is *legal* if color changes happen only at special vertices. That is, for any two consecutive edges $u{\to}v{\to}w$ in a legal walk, if the edges $u{\to}v$ and $v{\to}w$ have different colors, the intermediate vertex v must be special.

    Describe and analyze an algorithm that either returns the length of the shortest legal walk from vertex s to vertex t, or correctly reports that no such walk exists.[2]

    For example, if you are given the following graph below as input (where single arrows are red, double arrows are blue), with special vertices x and y, your algorithm should return the integer 8, which is the length of the shortest legal walk $s{\to}x{\to}a{\to}b{\to}x{\Rightarrow}y{\Rightarrow}b{\Rightarrow}c{\Rightarrow}t$. The shorter walk $s{\to}a{\to}b{\Rightarrow}c{\Rightarrow}t$ is not legal, because vertex b is not special.



11. ⟨⟨***S18***⟩⟩ Let G be a directed graph with weighted edges, in which every vertex is colored either red, green, or blue. Describe and analyze an algorithm to compute the length of the shortest walk in G that starts at a red vertex, then visits any number of vertices of any color, then visits a green vertex, then visits any number of vertices of any color, then visits a blue vertex, then visits any number of vertices of any color, and finally ends at a red vertex. Assume all edge weights are positive.

[2] If you've read China Miéville's excellent novel *The City & the City*, this problem should look familiar. If you haven't read *The City & the City*, I can't tell you why this problem should look familiar without spoiling the book.

CS/ECE 374 A ✧ Fall 2019

Fake Midterm 2

November 11, 2019

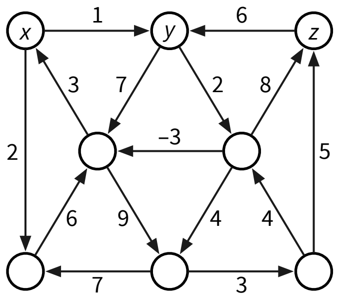
Real name:	
NetID:	

Gradescope name:	
Gradescope email:	

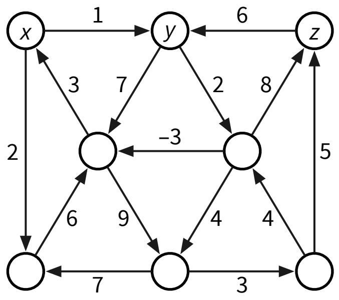
- ***Don't panic!***
- **All problems are described in more detail in a separate handout.** If any problem is unclear or ambiguous, please don't hesitate to ask us for clarification.
- If you brought anything except your writing implements, your **hand-written** double-sided 8½" × 11" cheat sheet, and your university ID, please put it away for the duration of the exam. In particular, please turn off and put away *all* medically unnecessary electronic devices.
- Please clearly print your real name, your university NetID, your Gradescope name, and your Gradescope email address in the boxes above. However, if you are using your real name and your university email address on Gradescope, you do ***not*** need to write everything twice. **We will not scan this page into Gradescope.**
- Please also print **only the name you are using on Gradescope** at the top of every page of the answer booklet, except this cover page. These are the pages we will scan into Gradescope.
- Please do not write outside the black boxes on each page; these indicate the area of the page that the scanner can actually see.
- If you run out of space for an answer, feel free to use the scratch pages at the back of the answer booklet, but **please clearly indicate where we should look**.
- Except for greedy algorithms, proofs are required for full credit if and only if we explicitly ask for them, using the word ***prove*** in bold italics.
- Please return ***all*** paper with your answer booklet: your question sheet, your cheat sheet, and all scratch paper.

CS/ECE 374 A ✧ Fall 2019 **Fake Midterm 2 Problem 1**	Gradescope name:

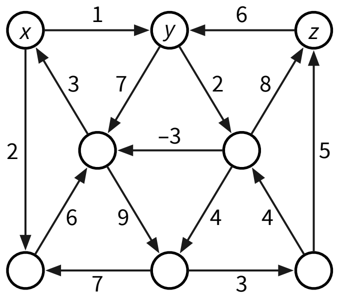
Clearly indicate the following structures in the directed graph below, or write NONE if the indicated structure does not exist. Don't be subtle; to indicate a collection of edges, draw a heavy black line along the entire length of each edge.



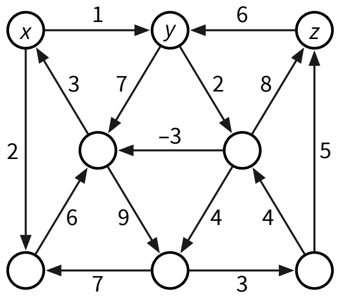
(a) A depth-first tree rooted at x.



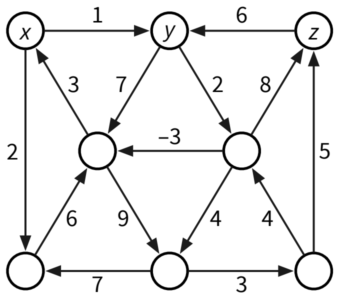
(b) A breadth-first tree rooted at y.



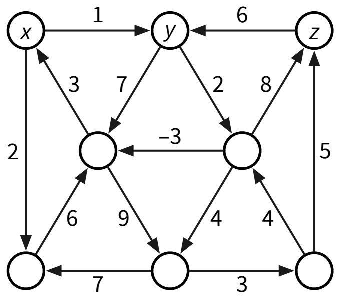
(c) A shortest-path tree rooted at z.



(d) The shortest directed cycle.



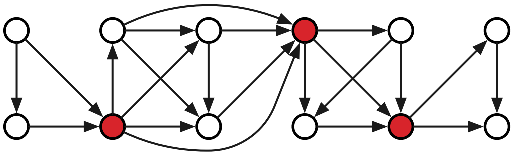
[scratch]



[scratch]

CS/ECE 374 A ✧ Fall 2019 **Fake Midterm 2 Problem 2**	Gradescope name:

A vertex v in a (weakly) connected graph G is called a ***cut vertex*** if the subgraph $G - v$ is disconnected. For example, the following graph has three cut vertices, which are shaded in the figure.



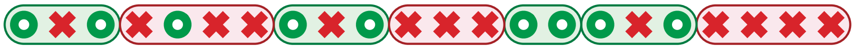
Suppose you are given a (weakly) connected *dag* G with one source and one sink. Describe and analyze an algorithm that returns True if G has a cut vertex and False otherwise.

CS/ECE 374 A ✧ Fall 2019 **Fake Midterm 2 Problem 3**	Gradescope name:

The City Council of Sham-Poobanana needs to partition Purple Street into voting districts. A total of n people live on Purple Street, at consecutive addresses $1, 2, \dots, n$. Each voting district must be a contiguous interval of addresses $i, i+1, \dots, j$ for some $1 \le i < j \le n$. By law, each Purple Street address must lie in exactly one district, and the number of addresses in each district must be between k and $2k$, where k is some positive integer parameter.

Every election in Sham-Poobanana is between two rival factions: Oceania and Eurasia. A majority of the City Council are from Oceania, so they consider a district to be *good* if more than half the residents of that district voted for Oceania in the previous election. Naturally, the City Council has complete voting records for all n residents.

For example, the figure below shows a legal partition of 22 addresses into 4 good districts and 3 bad districts, where $k = 2$. Each O indicates a vote for Oceania, and each X indicates a vote for Eurasia.



Describe an algorithm to find the largest possible number of *good* districts in a legal partition. Your input consists of the integer k and a boolean array GoodVote$[1\,..\,n]$ indicating which residents previously voted for Oceania (True) or Eurasia (False). You can assume that a legal partition exists. Analyze the running time of your algorithm in terms of the parameters n and k.

CS/ECE 374 A ✧ Fall 2019 **Fake Midterm 2 Problem 4**	Gradescope name:

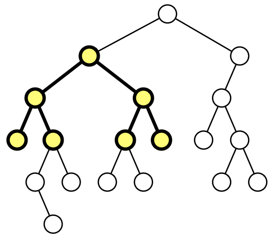
After graduation, you accept a job with Aviophiles-Я-Us, the leading traveling agency for people who love to fly. Your job is to build a system to help customers plan airplane trips from one city to another. Your customers love flying, but they absolutely despise airports. You know all the departure and arrival times of all the flights on the planet.

Suppose one of your customers wants to fly from city X to city Y. Describe an algorithm to find a sequence of flights that *minimizes the total time spent in airports*. Assume (unrealistically) that your customer can enter the starting airport immediately before the first flight leaves X, that they can leave the final airport at Y immediately after the final flight arrives at Y.

CS/ECE 374 A ✧ Fall 2019 Fake Midterm 2 Problem 5	Gradescope name:

For this problem, a *subtree* of a binary tree means any connected subgraph. A binary tree is *complete* if every internal node has two children, and every leaf has exactly the same depth.

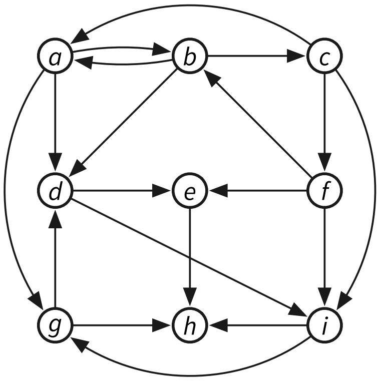
Describe and analyze a recursive algorithm to compute the ***largest complete subtree*** of a given binary tree. Your algorithm should return both the root and the depth of this subtree. For example, given the following tree T as input, your algorithm should return the left child of the root of T and the integer 2.



Write your answers in the separate answer booklet.

Please return this question sheet and your cheat sheet with your answers.

1. ***Clearly*** indicate the following structures in the directed graph below, or write NONE if the indicated structure does not exist. Don't be subtle; to indicate a collection of edges, draw a heavy black line along the entire length of each edge.



(a) A depth-first search tree rooted at vertex a.

(b) A breadth-first tree rooted at vertex a.

(c) The strong components of G. (Circle each strong component.)

(d) Draw the strong-component graph of G.

2. As the days get shorter in winter, Eggsy Hutmacher is increasingly worried about his walk home from work. The city has recently been invaded by the notorious Antimilliner gang, whose members hang out on dark street corners and steal hats from unwary passers-by, and a gentleman is simply *not* seen out in public without a hat. The city council is slowly installing street lamps at intersections to deter the Antimilliners, whose uncovered faces can be easily identified in the light. Eggsy keeps k extra hats in his briefcase in case of theft or other millinery emergencies.

   Eggsy has a map of the city in the form of an undirected graph G, whose vertices represent intersections and whose edges represent streets between them. A subset of the vertices are marked to indicate that the corresponding intersections are lit. Every edge e has a non-negative length $\ell(e)$. The graph has two special nodes s and t, which represent Eggsy's work and home, respectively.

   Describe an algorithm that computes the shortest path in G from s to t that visits at most k unlit vertices.

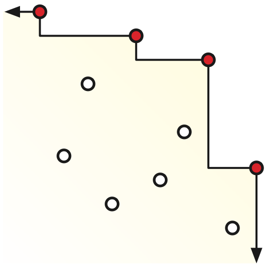
3. An undirected graph $G = (V, E)$ is ***bipartite*** if each of its vertices can be colored either black or white, so that every edge in E has one white endpoint and one black endpoint. Describe and analyze an algorithm to determine, given an undirected graph G as input, whether G is bipartite. *[Hint: Every tree is bipartite.]*

4. Satya is in charge of establishing a new testing center for the Standardized Awesomeness Test (SAT), and found an old conference hall that is perfect. The conference hall has n rooms of various sizes along a single long hallway, numbered in order from 1 through n. Satya knows exactly how many students fit into each room, and he wants to use a subset of the rooms to host as many students as possible for testing.

   Unfortunately, there have been several incidents of students cheating at other testing centers by tapping secret codes through walls. To prevent this type of cheating, Satya can use two adjacent rooms only if he demolishes the wall between them. The city's chief architect has determined that demolishing the walls on both sides of the same room would threaten the building's structural integrity. For this reason, Satya can never host students in three consecutive rooms.

   Describe an efficient algorithm that computes the largest number of students that Satya can host for testing without using three consecutive rooms. The input to your algorithm is an array $S[1\,..\,n]$, where each $S[i]$ is the (non-negative integer) number of students that can fit in room i.

5. Suppose you are given a set P of n points in the plane. A point $p \in P$ is *maximal* in P if no other point in P is both above and to the right of P. Intuitively, the maximal points define a "staircase" with all the other points of P below it.



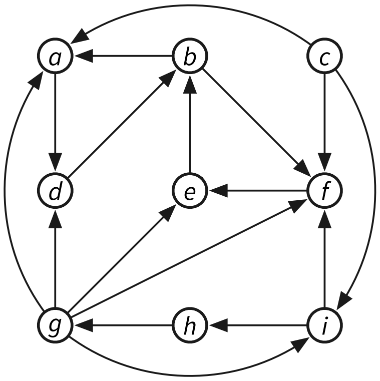
A set of ten points, four of which are maximal.

   Describe and analyze an algorithm to compute the number of maximal points in P in $O(n \log n)$ time. For example, given the ten points shown above, your algorithm should return the integer 4. The input to your algorithm is a pair of arrays $X[1\,..\,n]$ and $Y[1\,..\,n]$ containing the x- and y-coordinates of the points in P.

Write your answers in the separate answer booklet.

Please return this question sheet and your cheat sheet with your answers.

1. ***Clearly*** indicate the following structures in the directed graph below, or write NONE if the indicated structure does not exist. Don't be subtle; to indicate a collection of edges, draw a heavy black line along the entire length of each edge.



   (a) A depth-first search tree rooted at vertex c.

   (b) A breadth-first tree rooted at vertex c.

   (c) The strong components of G. (Circle each strong component.)

   (d) Draw the strong-component graph of G.

2. During her walk to work every morning, Rachel likes to buy a cappuccino at a local coffee shop, and a croissant at a local bakery. Her home town has *lots* of coffee shops and lots of bakeries, but strangely never in the same building. Punctuality is not Rachel's strongest trait, so to avoid losing her job, she wants to follow the shortest possible route.

   Rachel has a map of her home town in the form of an undirected graph G, whose vertices represent intersections and whose edges represent roads between them. A subset of the vertices are marked as bakeries; another disjoint subset of vertices are marked as coffee shops. The graph has two special nodes s and t, which represent Rachel's home and work, respectively.

   Describe an algorithm that computes the shortest path in G from s to t that visits both a bakery and a coffee shop, or correctly reports that no such path exists.

3. An undirected graph $G = (V, E)$ is ***bipartite*** if its vertices can be partitioned into two subsets L and R, such that every edge in E has one endpoint in L and one endpoint in R. Describe and analyze an algorithm to determine, given an undirected graph G as input, whether G is bipartite. *[Hint: Every tree is bipartite.]*

4. Satya is in charge of establishing a new testing center for the Standardized Awesomeness Test (SAT), and found an old conference hall that is perfect. The conference hall has n rooms of various sizes along a single long hallway, numbered in order from 1 through n. Each pair of adjacent rooms i and $i+1$ is separated by a single wall. Satya knows exactly how many students fit into each room, and he wants to use a subset of the rooms to host as many students as possible for testing.

   Unfortunately, there have been several incidents of students cheating at other testing centers by tapping secret codes through walls. To prevent this type of cheating, Satya can use two adjacent rooms only if he demolishes the wall between them. For example, if Satya wants to use rooms 1, 3, 4, 5, 7, 8, and 10, he must demolish three walls: between rooms 3 and 4, between rooms 4 and 5, and between rooms 7 and 8.

   The city's chief architect has determined that demolishing more than k walls would threaten the structural integrity of the building.

   Describe an efficient algorithm that computes the largest number of students that Satya can host for testing without demolishing more than k walls. The input to your algorithm is the integer k and an array $S[1\,..\,n]$, where each $S[i]$ is the (non-negative integer) number of students that can fit in room i.

5. Suppose you are given an array $A[1\,..\,n]$ of numbers.

   (a) Describe and analyze an algorithm that either returns two indices i and j such that $A[i]+A[j]=374$, or correctly reports that no such indices exist.

   (b) Describe and analyze an algorithm that either returns three indices i, j, and k such that $A[i]+A[j]+A[k]=374$, or correctly reports that no such indices exist.

   Do ***not*** use hashing. As always, faster correct algorithms are worth more points.

CS/ECE 374 A ✧ Fall 2019

Final Exam Study Questions

This is a "core dump" of potential questions for Midterm 1. This should give you a good idea of the *types* of questions that we will ask on the exam—in particular, there *will* be a series of True/False questions—but the actual exam questions may or may not appear in this handout. This list intentionally includes a few questions that are too long or difficult for exam conditions; these are indicated with a *star.

Don't forget to review the study problems for Midterms 1 and 2; the final exam is cumulative!

How to Use These Problems

Solving every problem in this handout is ***not*** the best way to study for the exam. Memorizing the solutions to every problem in this handout is the ***absolute worst*** way to study for the exam.

What we recommend instead is to work on a *sample* of the problems. Choose one or two problems at random from each section and try to solve them from scratch under exam conditions—by yourself, in a quiet room, with a 30-minute timer, *without* your notes, *without* the internet, and if possible, even without your cheat sheet. If you're comfortable solving a few problems in a particular section, you're probably ready for that type of problem on the exam. Move on to the next section.

Discussing problems with other people (in your study groups, in the review sessions, in office hours, or on Piazza) and/or looking up old solutions can be *extremely* helpful, but ***only after*** you have (1) made a good-faith effort to solve the problem on your own, and (2) you have either a candidate solution or some idea about where you're getting stuck.

If you find yourself getting stuck on a particular type of problem, try to figure out *why* you're stuck. Do you understand the problem statement? Are you stuck on choosing the right high-level approach? Are you stuck on the technical details? Or are you struggling to express your ideas clearly? (We *strongly* recommend writing solutions that follow the homework grading rubrics bullet-by-bullet.)

Similarly, if feedback from other people suggests that your solutions to a particular type of problem are incorrect or incomplete, try to figure out what you missed. For NP-hardness proofs: Are you choosing a good problem to reduce from? Are you reducing in the correct direction? Are you designing your reduction with both good instances and bad instances in mind? You're not trying *solve* the problem, are you? For undecidability proofs: Does the problem have the right structure to apply Rice's theorem? If you are arguing by reduction, are you reducing in the correct direction? You're not using *pronouns*, are you?

Remember that your goal is *not* merely to "understand" the solution to any particular problem, but to become more comfortable with solving a certain *type* of problem on your own. **"Understanding" is a trap; aim for mastery.** If you can identify specific steps that you find problematic, read more *about those steps*, focus your practice *on those steps*, and try to find helpful information *about those steps* to write on your cheat sheet. Then work on the next problem!

True or False? (All from previous final exams)

For each of the following questions, indicate ***every*** correct answer by marking the “Yes” box, and indicate ***every*** incorrect answer by marking the “No” box. **Assume *P* ≠ *NP*.** If there is any other ambiguity or uncertainty about an answer, mark the “No” box. For example:

Yes [✓ No] $x + y = 5$

Yes [✓ No] 3SAT can be solved in polynomial time.

[✓ Yes] No Jeff is not the Queen of England.

[✓ Yes] No If $P = NP$ then Jeff is the Queen of England.

The actual exam will include forty true/false questions. Each correct choice will be worth ½ point, each incorrect choice will be worth −¼ point, and each **IDK** will be worth +⅛ point.

---

1. Which of the following are a good English specifications of a recursive function that could possibly be used to compute the edit distance between two strings $A[1..n]$ and $B[1..n]$?

Yes No *Edit*(i, j) is the answer for i and j.

Yes No *Edit*(i, j) is the edit distance between $A[i]$ and $B[j]$.

Yes No

$$Edit[i,j] = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ Edit[i-1, j-1] & \text{if } A[i] = B[j] \\ \max\left\{\begin{matrix} 1 + Edit[i, j-1] \\ 1 + Edit[i-1, j] \\ 1 + Edit[i-1, j-1] \end{matrix}\right\} & \text{otherwise} \end{cases}$$

Yes No *Edit*$[1..n, 1..n]$ stores the edit distances for all prefixes.

Yes No *Edit*(i, j) is the edit distance between $A[i..n]$ and $B[j..n]$.

Yes No *Edit*$[i, j]$ is the value stored at row i and column j of the table.

Yes No *Edit*(i, j) is the edit distance between the last i characters of A and the last j characters of B.

Yes No *Edit*(i, j) is the edit distance when i and j are the current characters in A and B.

Yes No *Edit*(i, j, k, l) is the edit distance between substrings $A[i..j]$ and $B[k..l]$.

Yes No *[I don’t need an English description; my pseudocode is clear enough!]*

---

2. Which of the following statements are true for ***every*** language $L \subseteq \{\mathtt{0},\mathtt{1}\}^*$?

Yes No L is non-empty.

Yes No L is infinite.

Yes No L contains the empty string ε.

Yes No L^* is infinite.

Yes No L^* is regular.

Yes No L is accepted by some DFA if and only if L is accepted by some NFA.

Yes No L is described by some regular expression if and only if L is rejected by some NFA.

Yes No L is accepted by some DFA with 42 states if and only if L is accepted by some NFA with 42 states.

Yes No If L is decidable, then L is infinite.

Yes No If L is not decidable, then L is infinite.

Yes No If L is not regular, then L is undecidable.

Yes No If L has an infinite fooling set, then L is undecidable.

Yes No If L has a finite fooling set, then L is decidable.

Yes No If L is the union of two regular languages, then its complement $\overline{L}$ is regular.

Yes No If L is the union of two regular languages, then its complement $\overline{L}$ is context-free.

Yes No If L is the union of two decidable languages, then L is decidable.

Yes No If L is the union of two undecidable languages, then L is undecidable.

Yes No If $L \notin \mathrm{P}$, then L is not regular.

Yes No L is decidable if and only if its complement $\overline{L}$ is undecidable.

Yes No Both L and its complement $\overline{L}$ are decidable.

3. Which of the following statements are true for ***at least one*** language $L \subseteq \{\mathtt{0}, \mathtt{1}\}^*$?

Yes	No	L is non-empty.
Yes	No	L is infinite.
Yes	No	L contains the empty string ε.
Yes	No	L^* is finite.
Yes	No	L^* is not regular.
Yes	No	L is not regular but L^* is regular.
Yes	No	L is finite and L is undecidable.
Yes	No	L is decidable but L^* is not decidable.
Yes	No	L is not decidable but L^* is decidable.
Yes	No	L is the union of two decidable languages, but L is not decidable.
Yes	No	L is the union of two undecidable languages, but L is decidable.
Yes	No	L is accepted by an NFA with 374 states, but L is not accepted by a DFA with 374 states.
Yes	No	L is accepted by an DFA with 374 states, but L is not accepted by a NFA with 374 states.
Yes	No	L is regular and $L \notin \mathrm{P}$.
Yes	No	There is a Turing machine that accepts L.
Yes	No	There is an algorithm to decide whether an arbitrary given Turing machine accepts L.

4. Which of the following languages over the alphabet $\{\mathtt{0}, \mathtt{1}\}$ are ***regular***?

Yes	No	$\{\mathtt{0}^m\mathtt{1}^n \mid m \geq 0 \text{ and } n \geq 0\}$
Yes	No	All strings with the same number of 0s and 1s
Yes	No	Binary representations of all positive integers divisible by 17
Yes	No	Binary representations of all prime numbers less than 10^{100}
Yes	No	$\{ww \mid w \text{ is a palindrome}\}$

4. (continued) Which of the following languages over the alphabet $\{0,1\}$ are ***regular***?

Yes No $\{wxw \mid w \text{ is a palindrome and } x \in \{0,1\}^*\}$

Yes No $\{\langle M\rangle \mid M \text{ accepts a regular language}\}$

Yes No $\{\langle M\rangle \mid M \text{ accepts a finite number of non-palindromes}\}$

---

5. Which of the following languages/decision problems are ***decidable***?

Yes No $\varnothing$

Yes No $\{0^n 1^{2n} 0^n 1^{2n} \mid n \geq 0\}$

Yes No $\{ww \mid w \text{ is a palindrome}\}$

Yes No $\{\langle M\rangle \mid M \text{ accepts } \langle M\rangle \bullet \langle M\rangle\}$

Yes No $\{\langle M\rangle \mid M \text{ accepts a finite number of non-palindromes}\}$

Yes No $\{\langle M\rangle \bullet w \mid M \text{ accepts } ww\}$

Yes No $\{\langle M\rangle \bullet w \mid M \text{ accepts } ww \text{ after at most } |w|^2 \text{ transitions}\}$

Yes No Given an NFA N, is the language $L(N)$ infinite?

Yes No Given a context-free grammar G and a string w, is w in the language $L(G)$?

Yes No CircuitSat

Yes No Given an undirected graph G, does G contain a Hamiltonian cycle?

Yes No Given two Turing machines M and M', is there a string w that is accepted by both M and M'?

---

6. Which of the following languages can be proved undecidable ***using Rice's Theorem***?

Yes	No	$\varnothing$
Yes	No	$\{\mathtt{0}^n\mathtt{1}^{2n}\mathtt{0}^n\mathtt{1}^{2n} \mid n \geq 0\}$
Yes	No	$\{ww \mid w \text{ is a palindrome}\}$
Yes	No	$\{\langle M\rangle \mid M \text{ accepts an infinite number of strings}\}$
Yes	No	$\{\langle M\rangle \mid M \text{ accepts a finite number of strings}\}$
Yes	No	$\{\langle M\rangle \mid M \text{ accepts either } \langle M\rangle \text{ or } \langle M\rangle^R\}$
Yes	No	$\{\langle M\rangle \mid M \text{ accepts both } \langle M\rangle \text{ and } \langle M\rangle^R\}$
Yes	No	$\{\langle M\rangle \mid M \text{ does not accept exactly 374 palindromes}\}$
Yes	No	$\{\langle M\rangle \mid M \text{ accepts some string } w \text{ after at most } \lvert w\rvert^2 \text{ transitions}\}$
Yes	No	$\{\langle M\rangle\mathtt{\#}w \mid M \text{ rejects } w \text{ after at most } \lvert w\rvert^2 \text{ transitions}\}$
Yes	No	Given two Turing machines M and M', is there a string w that is accepted by both M and M'?

7. Recall the halting language $\textsc{Halt} = \{\langle M\rangle \bullet w \mid M \text{ halts on input } w\}$. Which of the following statements about its complement $\overline{\textsc{Halt}} = \Sigma^* \setminus \textsc{Halt}$ are true?

Yes	No	$\overline{\textsc{Halt}}$ is empty.
Yes	No	$\overline{\textsc{Halt}}$ is regular.
Yes	No	$\overline{\textsc{Halt}}$ is infinite.
Yes	No	$\overline{\textsc{Halt}}$ is decidable.
Yes	No	$\overline{\textsc{Halt}}$ is acceptable but not decidable.
Yes	No	$\overline{\textsc{Halt}}$ is not acceptable.

8. Suppose some language $A \in \{0,1\}^*$ reduces to another language $B \in \{0,1\}^*$. Which of the following statements ***must*** be true?

   Yes No A Turing machine that recognizes A can be used to construct a Turing machine that recognizes B.

   Yes No A is decidable.

   Yes No If B is decidable then A is decidable.

   Yes No If A is decidable then B is decidable.

   Yes No If B is NP-hard then A is NP-hard.

   Yes No If A has no polynomial-time algorithm then neither does B.

9. Suppose there is a ***polynomial-time*** reduction from problem A to problem B. Which of the following statements ***must*** be true?

   Yes No Problem B is NP-hard.

   Yes No A polynomial-time algorithm for B can be used to solve A in polynomial time.

   Yes No If B has no polynomial-time algorithm then neither does A.

   Yes No If A is NP-hard and B has a polynomial-time algorithm then $\mathrm{P} = \mathrm{NP}$.

   Yes No If B is NP-hard then A is NP-hard.

   Yes No If B is undecidable then A is undecidable.

10. Consider the following pair of languages:

- HamPath $:= \{G \mid G$ is an undirected graph with a Hamiltonian path$\}$
- Connected $:= \{G \mid G$ is a connected undirected graph$\}$

(For concreteness, assume that in both of these languages, graphs are represented by their adjacency matrices.) Which of the following ***must*** be true, assuming $P \neq NP$?

Yes	No	Connected $\in$ NP
Yes	No	HamPath $\in$ NP
Yes	No	HamPath is decidable.
Yes	No	There is no polynomial-time reduction from HamPath to Connected.
Yes	No	There is no polynomial-time reduction from Connected to HamPath.

11. Consider the following pair of languages:

- DirHamPath $:= \{G \mid G$ is a directed graph with a Hamiltonian path$\}$
- Acyclic $:= \{G \mid G$ is a directed acyclic graph$\}$

(For concreteness, assume that in both of these languages, graphs are represented by their adjacency matrices.) Which of the following ***must*** be true, assuming $P \neq NP$?

Yes	No	Acyclic $\in$ NP
Yes	No	Acyclic $\cap$ DirHamPath $\in$ P
Yes	No	DirHamPath is decidable.
Yes	No	There is a polynomial-time reduction from DirHamPath to Acyclic.
Yes	No	There is a polynomial-time reduction from Acyclic to DirHamPath.

12. Suppose we want to prove that the following language is undecidable.

$$\text{AlwaysHalts} := \big\{\langle M\rangle \bigm| M \text{ halts on every input string}\big\}$$

Rocket J. Squirrel suggests a reduction from the standard halting language

$$\text{Halt} := \big\{(\langle M\rangle, w) \bigm| M \text{ halts on inputs } w\big\}.$$

Specifically, given a Turing machine DecideAlwaysHalts that decides AlwaysHalts, Rocky claims that the following Turing machine DecideHalt decides Halt.

```
DecideHalt($\langle M\rangle, w$):
    Encode the following Turing machine $M'$:
        Bullwinkle($x$):
            if $M$ accepts $w$
                reject
            if $M$ rejects $w$
                accept
    return DecideAlwaysHalts($\langle$Bullwinkle$\rangle$)
```

Which of the following statements is true ***for all*** inputs $\langle M\rangle\#w$?

Yes	No	If M accepts w, then M' halts on every input string.
Yes	No	If M rejects w, then M' halts on every input string.
Yes	No	If M diverges on w, then M' halts on every input string.
Yes	No	If M accepts w, then DecideAlwaysHalts accepts $\langle$Bullwinkle$\rangle$.
Yes	No	If M rejects w, then DecideHalt rejects $(\langle M\rangle, w)$.
Yes	No	If M diverges on w, then DecideAlwaysHalts diverges on $\langle$Bullwinkle$\rangle$.
Yes	No	DecideHalt decides Halt. (That is, Rocky's reduction is correct.)

13. Suppose we want to prove that the following language is undecidable.

$$\textsc{Muggle} := \big\{\langle M\rangle \;\big|\; M \text{ accepts } \texttt{SCIENCE} \text{ but rejects } \texttt{MAGIC}\big\}$$

Professor Potter, your instructor in Defense Against Models of Computation and Other Dark Arts, suggests a reduction from the standard halting language

$$\textsc{Halt} := \big\{(\langle M\rangle, w) \;\big|\; M \text{ halts on inputs } w\big\}.$$

Specifically, suppose there is a Turing machine DetectoMuggletum that decides Muggle. Professor Potter claims that the following algorithm decides Halt.

```
DecideHalt($\langle M\rangle, w$):
  Encode the following Turing machine:
      RubberDuck($x$):
        run $M$ on input $w$
        if $x$ = MAGIC
            return False
        else
            return True
  return DetectoMuggletum($\langle$RubberDuck$\rangle$)
```

Which of the following statements is true ***for all*** inputs $\langle M\rangle\#w$?

Yes No If M accepts w, then RubberDuck accepts SCIENCE.

Yes No If M accepts w, then RubberDuck accepts CHOCOLATE.

Yes No If M rejects w, then RubberDuck rejects MAGIC.

Yes No If M rejects w, then RubberDuck halts on every input string.

Yes No If M diverges on w, then RubberDuck rejects every input string.

Yes No If M accepts w, then DetectoMuggletum accepts $\langle$RubberDuck$\rangle$.

Yes No If M rejects w, then DecideHalt rejects $(\langle M\rangle, w)$.

Yes No If M diverges on w, then DecideHalt rejects $(\langle M\rangle, w)$.

Yes No DecideHalt decides the language Halt. (That is, Professor Potter's reduction is actually correct.)

Yes No DecideHalt actually runs (or simulates) RubberDuck.

Yes No Muggle is decidable.

NP-hardness

1. A boolean formula is in *disjunctive normal form* (or *DNF*) if it consists of a *disjunction* (Or) or several *terms*, each of which is the conjunction (And) of one or more literals. For example, the formula

   $$(\overline{x} \wedge y \wedge \overline{z}) \vee (y \wedge z) \vee (x \wedge \overline{y} \wedge \overline{z})$$

   is in disjunctive normal form. DNF-Sat asks, given a boolean formula in disjunctive normal form, whether that formula is satisfiable.

   (a) Describe a polynomial-time algorithm to solve DNF-Sat.

   (b) What is the error in the following argument that P=NP?

   > *Suppose we are given a boolean formula in conjunctive normal form with at most three literals per clause, and we want to know if it is satisfiable. We can use the distributive law to construct an equivalent formula in disjunctive normal form. For example,*
   >
   > $$(x \vee y \vee \overline{z}) \wedge (\overline{x} \vee \overline{y}) \iff (x \wedge \overline{y}) \vee (y \wedge \overline{x}) \vee (\overline{z} \wedge \overline{x}) \vee (\overline{z} \wedge \overline{y})$$
   >
   > *Now we can use the algorithm from part (a) to determine, in polynomial time, whether the resulting DNF formula is satisfiable. We have just solved* 3Sat *in polynomial time. Since* 3Sat *is NP-hard, we must conclude that P=NP!*

2. A ***relaxed 3-coloring*** of a graph G assigns each vertex of G one of three colors (for example, red, green, and blue), such that ***at most one*** edge in G has both endpoints the same color.

   (a) Give an example of a graph that has a relaxed 3-coloring, but does not have a proper 3-coloring (where every edge has endpoints of different colors).

   (b) ***Prove*** that it is NP-hard to determine whether a given graph has a relaxed 3-coloring.

3. An ***ultra-Hamiltonian cycle*** in G is a closed walk C that visits every vertex of G exactly once, except for *at most one* vertex that C visits more than once.

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

   (b) ***Prove*** that it is NP-hard to determine whether a given graph contains a ultra-Hamiltonian cycle.

4. An ***infra-Hamiltonian cycle*** in G is a closed walk C that visits every vertex of G exactly once, except for *at most one* vertex that C does not visit at all.

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

   (b) ***Prove*** that it is NP-hard to determine whether a given graph contains a infra-Hamiltonian cycle.

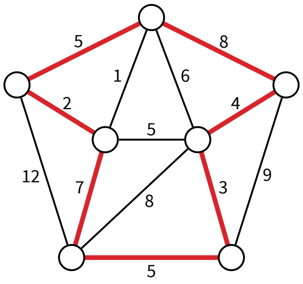
5. A ***quasi-satisfying assignment*** for a 3CNF boolean formula Φ is an assignment of truth values to the variables such that *at most one* clause in Φ does not contain a true literal. ***Prove*** that it is NP-hard to determine whether a given 3CNF boolean formula has a quasi-satisfying assignment.

6. A subset S of vertices in an undirected graph G is ***half-independent*** if each vertex in S is adjacent to *at most one* other vertex in S. Prove that finding the size of the largest half-independent set of vertices in a given undirected graph is NP-hard.

7. A subset S of vertices in an undirected graph G is ***sort-of-independent*** if if each vertex in S is adjacent to *at most 374* other vertices in S. Prove that finding the size of the largest sort-of-independent set of vertices in a given undirected graph is NP-hard.

8. A subset S of vertices in an undirected graph G is ***almost independent*** if at most 374 edges in G have both endpoints in S. Prove that finding the size of the largest almost-independent set of vertices in a given undirected graph is NP-hard.

9. Let G be an undirected graph with weighted edges. A ***heavy Hamiltonian cycle*** is a cycle C that passes through each vertex of G exactly once, such that the total weight of the edges in C is more than half of the total weight of all edges in G. Prove that deciding whether a graph has a heavy Hamiltonian cycle is NP-hard.



A heavy Hamiltonian cycle. The cycle has total weight 34; the graph has total weight 67.

10. (a) A ***tonian path*** in a graph G is a path that goes through at least half of the vertices of G. Show that determining whether a graph has a tonian path is NP-hard.

    (b) A ***tonian cycle*** in a graph G is a cycle that goes through at least half of the vertices of G. Show that determining whether a graph has a tonian cycle is NP-hard. *[Hint: Use part (a). Or not.]*

11. Prove that the following variants of SAT is NP-hard. *[Hint: Describe reductions from 3SAT.]*

    (a) Given a boolean formula Φ in conjunctive normal form, where *each variable appears in at most three clauses*, determine whether F has a satisfying assignment. *[Hint: First consider the variant where each variable appears in at most* ***five*** *clauses.]*

    (b) Given a boolean formula Φ in conjunctive normal form *and given one satisfying assignment for* Φ, determine whether Φ has at least one other satisfying assignment.

12. Jerry Springer and Maury Povich have decided not to compete with each other over scheduling guests during the next talk-show season. There is only one set of Weird People who either host would consider having on their show. The hosts want to divide the Weird People into two (disjoint) groups: those to appear on Jerry's show, and those to appear on Maury's show. (Neither wants to "recycle" a guest that appeared on the other's show.)

Both Jerry and Maury have preferences about which Weird People they are particularly interested in. For example, Jerry wants to be sure to get at least one person who fits the category "had extra-terrestrial affair". Thus, on his list of preferences, he writes "w_1 or w_3 or w_{45}", since weird people numbered 1, 3, and 45 are the only ones who fit that description. Jerry has other preferences as well, so he lists those also. Similarly, Maury might like to guarantee that his show includes at least one guest who confesses to "really enjoying Rice's theorem". Each potential guest may fall into any number of different categories, such as the person who enjoys Rice's theorem more than the extra-terrestrial affair they had.

Jerry and Maury each prepare a list reflecting all of their preferences. Each list contains a collection of statements of the form "(w_i or w_j or w_k)". Your task is to prove that it is NP-hard to find an assignment of weird guests to the two shows that satisfies all of Jerry's preferences and all of Maury's preferences.

(a) The problem NoMixedClauses3Sat is the special case of 3Sat where the input formula cannot contain a clause with both a negated variable and a non-negated variable. Prove that NoMixedClauses3Sat is NP-hard. *[Hint: Reduce from the standard 3Sat problem.]*

(b) Describe a polynomial-time reduction from NoMixedClauses3Sat to 3Sat.

13. The president of Sham-Poobanana University is planning a Mardi Gras party for the university staff. His staff has a hierarchical structure; that is, the supervisor relation forms a directed, acyclic graph, with the president as the only source, and there is an edge from person i to person j in the graph if and only if person i is an immediate supervisor of person j. (Many people on the staff have multiple positions, and thus have several immediate supervisors.) In order to make the party fun for all guests, the president wants to ensure that if a person i attends, then none of i's immediate supervisors can attend.

    By mining each staff member's email and social media accounts, Sham-Poobanana University Human Resources has determined a "party-hound" rating for each staff member, which is a non-negative real number reflecting how likely it is that the person will leave the party wearing a monkey suit and a lampshade.

    Show that it is NP-hard to determine a guest-list that *maximizes* the sum of the party-hound ratings of all invited guests, subject to the supervisor constraint.

    [Hint: This problem can be solved in polynomial time when the input graph is a tree!]

14. Prove that the following problem (which we call Match) is NP-hard. The input is a finite set S of strings, all of the same length n, over the alphabet $\{0, 1, 2\}$. The problem is to determine whether there is a string $w \in \{0, 1\}^n$ such that for every string $s \in S$, the strings s and w have the same symbol in at least one position.

    For example, given the set $S = \{01220, 21110, 21120, 00211, 11101\}$, the correct output is True, because the string $w = 01001$ matches the first three strings of S in the second position, and matches the last two strings of S in the last position. On the other hand, given the set $S = \{00, 11, 01, 10\}$, the correct output is False.

    [Hint: Describe a reduction from SAT (or 3SAT)]

15. To celebrate the end of the semester, Professor Jarling want to treat himself to an ice-cream cone, at the *Polynomial House of Flavors*. For a fixed price, he can build a cone with as many

scoops as he'd like. Because he has good balance (and because we want this problem to work out), assume that he can balance any number of scoops on top of the cone without it tipping over. He plans to eat the ice cream one scoop at a time, from top to bottom, and doesn't want more than one scoop of any flavor.

However, he realizes that eating a scoop of bubblegum ice cream immediately after the scoop of potatoes-and-gravy ice cream would be unpalatable; these two flavors clearly should not be placed next to each other in the stack. He has other similar constraints; certain pairs of flavors cannot be adjacent in the stack.

He'd like to get as much ice cream as he can for the one fee by building the tallest cone possible that meets his flavor-incompatibility constraints. Prove that this problem is NP-hard.

16. Prove that the following problems are NP-hard.

    (a) Given an undirected graph G, does G contain a simple path that visits all but 17 vertices?

    (b) Given an undirected graph G, does G have a spanning tree in which every node has degree at most 23?

    (c) Given an undirected graph G, does G have a spanning tree with at most 42 leaves?

17. Prove that the following problems are NP-hard.

    (a) Given an undirected graph G, is it possible to color the vertices of G with three different colors, so that at most 31337 edges have both endpoints the same color?

    (b) Given an undirected graph G, is it possible to color the vertices of G with three different colors, so that each vertex has at most 8675309 neighbors with the same color?

18. At the end of every semester, Jeff needs to solve the following ExamDesign problem. He has a list of problems, and he knows for each problem which students will *really enjoy* that problem. He needs to choose a subset of problems for the exam such that for each student in the class, the exam includes at least one question that student will really enjoy. On the other hand, he does not want to spend the entire summer grading an exam with dozens of questions, so the exam must also contain as few questions as possible. Prove that the ExamDesign problem is NP-hard.

19. Which of the following results would resolve the P vs. NP question? Justify each answer with a short sentence or two.

    (a) The construction of a polynomial time algorithm for some problem in NP.

    (b) A polynomial-time reduction from 3Sat to the language $\{\mathtt{0}^n\mathtt{1}^n \mid n \geq 0\}$.

    (c) A polynomial-time reduction from $\{\mathtt{0}^n\mathtt{1}^n \mid n \geq 0\}$ to 3Sat.

    (d) A polynomial-time reduction from 3Color to MinVertexCover.

    (e) The construction of a nondeterministic Turing machine that cannot be simulated by any deterministic Turing machine with the same running time.

Some useful NP-hard problems. You are welcome to use any of these in your own NP-hardness proofs, except of course for the specific problem you are trying to prove NP-hard.

CircuitSat: Given a boolean circuit, are there any input values that make the circuit output True?

3Sat: Given a boolean formula in conjunctive normal form, with exactly three distinct literals per clause, does the formula have a satisfying assignment?

MaxIndependentSet: Given an undirected graph G, what is the size of the largest subset of vertices in G that have no edges among them?

MaxClique: Given an undirected graph G, what is the size of the largest complete subgraph of G?

MinVertexCover: Given an undirected graph G, what is the size of the smallest subset of vertices that touch every edge in G?

MinSetCover: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subcollection whose union is S?

MinHittingSet: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subset of S that intersects every subset S_i?

3Color: Given an undirected graph G, can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

HamiltonianPath: Given graph G (either directed or undirected), is there a path in G that visits every vertex exactly once?

HamiltonianCycle: Given a graph G (either directed or undirected), is there a cycle in G that visits every vertex exactly once?

TravelingSalesman: Given a graph G (either directed or undirected) with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in G?

LongestPath: Given a graph G (either directed or undirected, possibly with weighted edges), what is the length of the longest simple path in G?

SteinerTree: Given an undirected graph G with some of the vertices marked, what is the minimum number of edges in a subtree of G that contains every marked vertex?

SubsetSum: Given a set X of positive integers and an integer k, does X have a subset whose elements sum to k?

Partition: Given a set X of positive integers, can X be partitioned into two subsets with the same sum?

3Partition: Given a set X of $3n$ positive integers, can X be partitioned into n three-element subsets, all with the same sum?

IntegerLinearProgramming: Given a matrix $A \in \mathbb{Z}^{n \times d}$ and two vectors $b \in \mathbb{Z}^n$ and $c \in Z^d$, compute $\max\{c \cdot x \mid Ax \leq b, x \geq 0, x \in \mathbb{Z}^d\}$.

FeasibleILP: Given a matrix $A \in \mathbb{Z}^{n \times d}$ and a vector $b \in \mathbb{Z}^n$, determine whether the set of feasible integer points $\max\{x \in \mathbb{Z}^d \mid Ax \leq b, x \geq 0\}$ is empty.

Draughts: Given an $n \times n$ international draughts configuration, what is the largest number of pieces that can (and therefore must) be captured in a single move?

SuperMarioBrothers: Given an $n \times n$ Super Mario Brothers level, can Mario reach the castle?

SteamedHams: Aurora borealis? At this time of year, at this time of day, in this part of the country, localized entirely within your kitchen? May I see it?

Turing Machines and Undecidability

The only undecidability questions on this semester's final exam will be True/False, but the following problems might still be useful as practice.

For each of the following languages, either ***sketch*** an algorithm to decide that language or ***prove*** that the language is undecidable, using a diagonalization argument, a reduction argument, Rice's theorem, closure properties, or some combination of the above. Recall that w^R denotes the reversal of string w.

1. $\varnothing$
2. $\{\mathtt{0}^n\mathtt{1}^n\mathtt{2}^n \mid n \ge 0\}$
3. $\{A \in \{\mathtt{0},\mathtt{1}\}^{n\times n} \mid n \ge 0 \text{ and } A \text{ is the adjacency matrix of a dag with } n \text{ vertices}\}$
4. $\{A \in \{\mathtt{0},\mathtt{1}\}^{n\times n} \mid n \ge 0 \text{ and } A \text{ is the adjacency matrix of a 3-colorable graph with } n \text{ vertices}\}$
5. $\{\langle M\rangle \mid M \text{ accepts } \langle M\rangle^R\}$
6. $\{\langle M\rangle \mid M \text{ accepts } \langle M\rangle^R\} \cap \{\langle M\rangle \mid M \text{ rejects } \langle M\rangle^R\}$
7. $\{\langle M\rangle\mathtt{\#}w \mid M \text{ accepts } ww^R\}$
8. $\{\langle M\rangle \mid M \text{ accepts } \mathtt{RICESTHEOREM}\}$
9. $\{\langle M\rangle \mid M \text{ rejects } \mathtt{RICESTHEOREM}\}$
10. $\{\langle M\rangle \mid M \text{ accepts at least one palindrome}\}$
11. $\Sigma^* \setminus \{\langle M\rangle \mid M \text{ accepts at least one palindrome}\}$
12. $\{\langle M\rangle \mid M \text{ rejects at least one palindrome}\}$
13. $\{\langle M\rangle \mid M \text{ accepts exactly one string of length } \ell\text{, for each integer } \ell \ge 0\}$
14. $\{\langle M\rangle \mid \textsc{Accept}(M) \text{ has an infinite fooling set}\}$
15. $\{\langle M\rangle\mathtt{\#}\langle M'\rangle \mid \textsc{Accept}(M) \cap \textsc{Accept}(M') \ne \varnothing\}$
16. $\{\langle M\rangle\mathtt{\#}\langle M'\rangle \mid \textsc{Accept}(M) \oplus \textsc{Reject}(M') \ne \varnothing\}$ — Here $\oplus$ means exclusive-or.

Some useful undecidable problems. You are welcome to use any of these in your own undecidability proofs, except of course for the specific problem you are trying to prove undecidable.

$$
\begin{aligned}
\textsc{SelfReject} &:= \big\{\langle M\rangle \bigm| M \text{ rejects } \langle M\rangle\big\}\\
\textsc{SelfAccept} &:= \big\{\langle M\rangle \bigm| M \text{ accepts } \langle M\rangle\big\}\\
\textsc{SelfHalt} &:= \big\{\langle M\rangle \bigm| M \text{ halts on } \langle M\rangle\big\}\\
\textsc{SelfDiverge} &:= \big\{\langle M\rangle \bigm| M \text{ does not halt on } \langle M\rangle\big\}\\
\textsc{Reject} &:= \big\{\langle M\rangle\#w \bigm| M \text{ rejects } w\big\}\\
\textsc{Accept} &:= \big\{\langle M\rangle\#w \bigm| M \text{ accepts } w\big\}\\
\textsc{Halt} &:= \big\{\langle M\rangle\#w \bigm| M \text{ halts on } w\big\}\\
\textsc{Diverge} &:= \big\{\langle M\rangle\#w \bigm| M \text{ does not halt on } w\big\}\\
\textsc{NeverReject} &:= \big\{\langle M\rangle \bigm| \textsc{Reject}(M) = \varnothing\big\}\\
\textsc{NeverAccept} &:= \big\{\langle M\rangle \bigm| \textsc{Accept}(M) = \varnothing\big\}\\
\textsc{NeverHalt} &:= \big\{\langle M\rangle \bigm| \textsc{Halt}(M) = \varnothing\big\}\\
\textsc{NeverDiverge} &:= \big\{\langle M\rangle \bigm| \textsc{Diverge}(M) = \varnothing\big\}
\end{aligned}
$$

CS/ECE 374 A ✧ Fall 2019

Final Exam

December 13, 2019

Real name:	
NetID:	

Gradescope name:	
Gradescope email:	

- ***Don't panic!***

- If you brought anything except your writing implements and your two double-sided 8½" × 11" cheat sheets, please put it away for the duration of the exam. In particular, please turn off and put away *all* medically unnecessary electronic devices.

- Please clearly print your real name, your university NetID, your Gradescope name, and your Gradescope email address in the boxes above. **We will not scan this page into Gradescope.**

- Please also print **only the name you are using on Gradescope** at the top of every page of the answer booklet, except this cover page. These are the pages we will scan into Gradescope.

- Please do not write outside the black boxes on each page; these indicate the area of the page that the scanner can actually see.

- **Please read the entire exam before writing anything.** Please ask for clarification if any question is unclear.

- **The exam lasts 180 minutes.**

- If you run out of space for an answer, continue on the back of the page, or on the blank pages at the end of this booklet, **but please tell us where to look.** Alternatively, feel free to tear out the blank pages and use them as scratch paper.

- As usual, answering any (sub)problem with "I don't know" (and nothing else) is worth 25% partial credit. **Yes, even for problem 1.** Correct, complete, but suboptimal solutions are *always* worth more than 25%. A blank answer is not the same as "I don't know".

- **Please return your cheat sheets and all scratch paper with your answer booklet.**

- May the Sith be with you.

Beware of the man who works hard to learn something,
learns it, and finds himself no wiser than before.

He is full of murderous resentment of people who are ignorant
without having come by their ignorance the hard way.

— Bokonon

CS/ECE 374 A ✧ Fall 2019 **Final Exam Problem 1**	Gradescope name:

For each of the following questions, indicate ***every*** correct answer by marking the "Yes" box, and indicate ***every*** incorrect answer by marking the "No" box. **Assume P ≠ NP.** If there is any other ambiguity or uncertainty, mark the "No" box. For example:

□ Yes ☒ No □ IDK $x + y = 5$

□ Yes ☒ No □ IDK 3SAT can be solved in polynomial time.

☒ Yes □ No □ IDK Jeff is not the Queen of England.

☒ Yes □ No □ IDK If $P = NP$ then Jeff is the Queen of England.

There are **40** yes/no choices altogether. Each correct choice is worth +½ point; each incorrect choice is worth −¼ point; each checked "IDK" is worth +1/8 point.

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

□ Yes □ No □ IDK L^* is non-empty.

□ Yes □ No □ IDK L^* is regular.

□ Yes □ No □ IDK L^* is decidable.

□ Yes □ No □ IDK If L is NP-hard, then L is not regular.

□ Yes □ No □ IDK If L is not regular, then L is undecidable.

□ Yes □ No □ IDK If L is context-free, then L is infinite.

□ Yes □ No □ IDK L is the intersection of two regular languages if and only if L is regular.

□ Yes □ No □ IDK L is decidable if and only if L^* is decidable.

□ Yes □ No □ IDK L is decidable if and only if its reversal $L^R = \{w^R \mid w \in L\}$ is decidable. (Recall that w^R denotes the reversal of the string w.)

□ Yes □ No □ IDK L is decidable if and only if its complement $\overline{L}$ is undecidable.

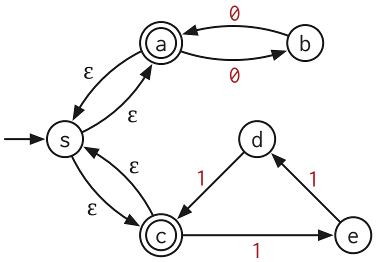
(b) Consider the following sets of undirected graphs:

- Trees is the set of all connected undirected graphs with no cycles.
- 3Color is the set of all undirected graphs that can be properly colored using at most 3 colors.

(For concreteness, assume that in both of these languages, graphs are represented by their adjacency matrices.) Which of the following ***must*** be true, assuming P≠NP?

Yes	No	IDK	Trees $\in NP$
Yes	No	IDK	Trees $\subseteq$ 3Color
Yes	No	IDK	There is a polynomial-time reduction from Trees to 3Color
Yes	No	IDK	There is a polynomial-time reduction from 3Color to Trees
Yes	No	IDK	Trees is NP-hard.

(c) Let M be the following NFA:



Which of the following statements about M are true?

Yes	No	IDK	M accepts the empty string ε
Yes	No	IDK	$\delta^*(s, 010) = \{s, a, c\}$
Yes	No	IDK	ε-reach$(a) = \{s, a, c\}$
Yes	No	IDK	M rejects the string 11100111000
Yes	No	IDK	$L(M) = (00)^* + (111)^*$

1 (continued)

(d) Which of the following languages over the alphabet $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ are ***regular***? Recall that $\#(a, w)$ denotes the number of times symbol a appears in string w.

Yes	No	IDK	The intersection of two regular languages
Yes	No	IDK	$\left\{w \in \Sigma^* \;\middle|\; \lvert w\rvert \text{ is prime}\right\}$
Yes	No	IDK	$\{w \in \Sigma^* \mid \#(\mathtt{0}, w) + \#(\mathtt{1}, w) > 374\}$
Yes	No	IDK	$\{w \in \Sigma^* \mid \#(\mathtt{0}, w) - \#(\mathtt{1}, w) > 374\}$
Yes	No	IDK	The language generated by the context-free grammar $S \rightarrow \mathtt{0}S \mid \mathtt{10}S \mid \varepsilon$

(e) Which of the following languages or problems are ***decidable***?

Yes	No	IDK	Σ^*
Yes	No	IDK	3Sat
Yes	No	IDK	$\{\langle M\rangle \mid M$ accepts every string whose length is prime$\}$
Yes	No	IDK	$\{\langle M\rangle \mid M$ accepts all strings in $\mathtt{0}^*$ and rejects all strings in $\mathtt{1}^*\}$
Yes	No	IDK	$\{\langle M\rangle \mid M$ is a Turing machine with at least two states$\}$

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

Yes	No	IDK	Σ^*
Yes	No	IDK	3Sat
Yes	No	IDK	$\{\langle M\rangle \mid M$ accepts every string whose length is prime$\}$
Yes	No	IDK	$\{\langle M\rangle \mid M$ accepts all strings in $\mathtt{0}^*$ and rejects all strings in $\mathtt{1}^*\}$
Yes	No	IDK	$\{\langle M\rangle \mid M$ is a Turing machine with at least two states$\}$

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

$$\textsc{Marvin} := \left\{\langle M\rangle \;\middle|\; M \text{ rejects an infinite number of strings}\right\}$$

Professor Prefect, your instructor in Vogon Poetry and Knowing Where Your Towel Is, suggests a reduction from the standard halting language

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

Specifically, suppose there is a program ParanoidAndroid that decides Marvin. Professor Prefect claims that the following algorithm decides Halt.

```
DecideHalt($(\langle M\rangle, w)$):
  Write code for the following algorithm:
      HoopyFrood($x$):
        run $M$ on input $w$
        if $x$ = DONTPANIC
            return True
        else
            return False
  return ParanoidAndroid($\langle$HoopyFrood$\rangle$)
```

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

Yes	No	IDK	If M accepts w, then HoopyFrood accepts BEEBLEBROX.
Yes	No	IDK	If M rejects w, then HoopyFrood rejects BEEBLEBROX.
Yes	No	IDK	If M hangs on w, then HoopyFrood rejects DONTPANIC.
Yes	No	IDK	ParanoidAndroid accepts ⟨HoopyFrood⟩.
Yes	No	IDK	DecideHalt decides Halt; that is, Professor Prefect's proof is correct.

CS/ECE 374 A ✧ Fall 2019 **Final Exam Problem 2**	Gradescope name:

You are planning a hiking trip in Jellystone National Park over winter break. You have a complete map of the park's trails; the map indicates that some trail segments have a high risk of bear encounters. All visitors to the park are required to purchase a canister of bear repellent. You can safely traverse a high-bear-risk trail segment only by *completely* using up a *full* canister of bear repellent. The park rangers have installed refilling stations at several locations around the park, where you can refill empty canisters at no cost. The canisters themselves are expensive and heavy, so you cannot carry more than one. Because the trails are narrow, each trail segment allows traffic in only one direction.

You have converted the trail map into a directed graph $G = (V, E)$, whose vertices represent trail intersections, and whose edges represent trail segments. A subset $R \subseteq V$ of the vertices indicate the locations of the *R*epellent *R*efilling stations, and a subset $B \subseteq E$ of the edges are marked as having a high risk of *B*ears. Your campsite appears on the map as a particular vertex $s \in V$, and the visitor center is another vertex $t \in V$.

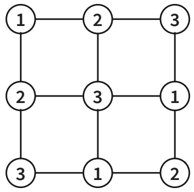
(a) Describe and analyze an algorithm to decide if you can safely walk from your campsite s to the visitor center t. Assume there is a refill station at your camp site, and another refill station at the visitor center.

(b) Describe and analyze an algorithm to decide if you can walk safely from any refill station any other refill station. In other words, for *every* pair of vertices u and v in R, is there a safe path from u to v?

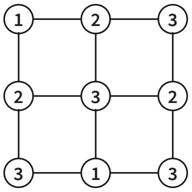
2 (continued)

CS/ECE 374 A ✧ Fall 2019 **Final Exam Problem 3**	Gradescope name:

Recall that a *proper 3-coloring* of a graph G assigns each vertex of G one of three colors, so that every edge of G has endpoints with different colors. A proper 3-coloring is *balanced* if each color is assigned to *exactly* the same number of vertices.



A balanced proper 3-coloring.



A proper 3-coloring that is not balanced.

The Balanced3Color problem asks, given an undirected graph G, whether G has a balanced proper 3-coloring. ***Prove*** that Balanced3Color is NP-hard.

CS/ECE 374 A ✧ Fall 2019 **Final Exam Problem 4**	Gradescope name:

For each of the following languages, state whether the language is regular or not, and then justify your answer as follows:

- If the language is regular, ***either*** give an regular expression that describes the language, ***or*** draw/describe a DFA or NFA that accepts the language. You do not need to prove that your automaton or regular expression is correct.
- If the language is not regular, ***prove*** that the language is not regular.

[Hint: Exactly one of these languages is regular.]

(a) $\{xy \mid x, y \in \Sigma^+ \text{ and } x \text{ and } y \text{ are both palindromes}\}$

(b) $\{xy \mid x, y \in \Sigma^+ \text{ and } x \text{ is not a palindrome}\}$

CS/ECE 374 A ✧ Fall 2019 **Final Exam Problem 5**	Gradescope name:

(a) Recall that a *palindrome* is any string that is equal to its reversal, like `REDIVIDER` or `POOP`. Describe an algorithm to find the length of the longest subsequence of a given string that is a palindrome.

(b) A *double palindrome* is the concatenation of two *non-empty* palindromes, like `POOPREDIVIDER` or `POOPPOOP`. Describe an algorithm to find the length of the longest subsequence of a given string that is a *double* palindrome. *[Hint: Use your algorithm from part (a).]*

For both algorithms, the input is an array $A[1..n]$, and the output is an integer. For example, given the string `MAYBEDYNAMICPROGRAMMING` as input, your algorithm for part (a) should return 7 (for the palindrome subsequence `NMRORMN`), and your algorithm for part (b) should return 12 (for the double palindrome subsequence `MAYBYAMIRORI`).

CS/ECE 374 A ✧ Fall 2019 **Final Exam Problem 6**	Gradescope name:

Let M be an arbitrary DFA. Describe and analyze an efficient algorithm to decide whether M rejects an infinite number of strings.

(scratch paper)

(scratch paper)

(scratch paper)

Some useful NP-hard problems. You are welcome to use any of these in your own NP-hardness proofs, except of course for the specific problem you are trying to prove NP-hard.

CircuitSat: Given a boolean circuit, are there any input values that make the circuit output True?

3Sat: Given a boolean formula in conjunctive normal form, with exactly three distinct literals per clause, does the formula have a satisfying assignment?

MaxIndependentSet: Given an undirected graph G, what is the size of the largest subset of vertices in G that have no edges among them?

MaxClique: Given an undirected graph G, what is the size of the largest complete subgraph of G?

MinVertexCover: Given an undirected graph G, what is the size of the smallest subset of vertices that touch every edge in G?

MinSetCover: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subcollection whose union is S?

MinHittingSet: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subset of S that intersects every subset S_i?

3Color: Given an undirected graph G, can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

HamiltonianPath: Given graph G (either directed or undirected), is there a path in G that visits every vertex exactly once?

HamiltonianCycle: Given a graph G (either directed or undirected), is there a cycle in G that visits every vertex exactly once?

TravelingSalesman: Given a graph G (either directed or undirected) with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in G?

LongestPath: Given a graph G (either directed or undirected, possibly with weighted edges), what is the length of the longest simple path in G?

SteinerTree: Given an undirected graph G with some of the vertices marked, what is the minimum number of edges in a subtree of G that contains every marked vertex?

SubsetSum: Given a set X of positive integers and an integer k, does X have a subset whose elements sum to k?

Partition: Given a set X of positive integers, can X be partitioned into two subsets with the same sum?

3Partition: Given a set X of $3n$ positive integers, can X be partitioned into n three-element subsets, all with the same sum?

IntegerLinearProgramming: Given a matrix $A \in \mathbb{Z}^{n \times d}$ and two vectors $b \in \mathbb{Z}^n$ and $c \in Z^d$, compute $\max\{c \cdot x \mid Ax \leq b, x \geq 0, x \in \mathbb{Z}^d\}$.

FeasibleILP: Given a matrix $A \in \mathbb{Z}^{n \times d}$ and a vector $b \in \mathbb{Z}^n$, determine whether the set of feasible integer points $\max\{x \in \mathbb{Z}^d \mid Ax \leq b, x \geq 0\}$ is empty.

Draughts: Given an $n \times n$ international draughts configuration, what is the largest number of pieces that can (and therefore must) be captured in a single move?

SuperMarioBrothers: Given an $n \times n$ Super Mario Brothers level, can Mario reach the castle?

SteamedHams: Aurora borealis? At this time of year, at this time of day, in this part of the country, localized entirely within your kitchen? May I see it?

CS/ECE 374 A ✧ Fall 2019

Final Exam

December 16, 2019

Real name:	
NetID:	

Gradescope name:	
Gradescope email:	

- ***Don't panic!***
- If you brought anything except your writing implements and your two double-sided 8½" × 11" cheat sheets, please put it away for the duration of the exam. In particular, please turn off and put away *all* medically unnecessary electronic devices.
- Please clearly print your real name, your university NetID, your Gradescope name, and your Gradescope email address in the boxes above. **We will not scan this page into Gradescope.**
- **Print the name you are using on Gradescope** at the top of every page of the answer booklet, except this cover page. These are the pages we will scan into Gradescope.
- Please do not write outside the black boxes on each page; these indicate the area of the page that the scanner can actually see.
- **Please read the entire exam before writing anything.** Please ask for clarification if any question is unclear.
- **The exam lasts 180 minutes.**
- If you run out of space for an answer, continue on the back of the page, or on the blank pages at the end of this booklet, **but please tell us where to look.** Alternatively, feel free to tear out the blank pages and use them as scratch paper.
- As usual, answering any (sub)problem with "I don't know" (and nothing else) is worth 25% partial credit. **Yes, even for problem 1.** Correct, complete, but suboptimal solutions are *always* worth more than 25%. A blank answer is not the same as "I don't know".
- **Please return your cheat sheets and all scratch paper with your answer booklet.**
- Good luck, and thanks for a great semester!

Beware of the man who works hard to learn something,
learns it, and finds himself no wiser than before.

He is full of murderous resentment of people who are ignorant
without having come by their ignorance the hard way.

— Bokonon

CS/ECE 374 A ✧ Fall 2019 **Final Exam Problem 1**	Gradescope name:

For each of the following questions, indicate ***every*** correct answer by marking the "Yes" box, and indicate ***every*** incorrect answer by marking the "No" box. **Assume P $\neq$ NP.** If there is any other ambiguity or uncertainty, mark the "No" box. For example:

Yes [✓ No] IDK — $x + y = 5$

Yes [✓ No] IDK — 3SAT can be solved in polynomial time.

[✓ Yes] No IDK — Jeff is not the Queen of England.

[✓ Yes] No IDK — If $P = NP$ then Jeff is the Queen of England.

There are **40** yes/no choices altogether. Each correct choice is worth +½ point; each incorrect choice is worth −¼ point; each checked "IDK" is worth +1/8 point.

(a) Which of the following statements are true for ***at least one*** language $L \subseteq \{0, 1\}^*$?

Yes No IDK — L^* is empty.

Yes No IDK — L^* is not regular.

Yes No IDK — L^* is decidable.

Yes No IDK — L is decidable but L^* is undecidable.

Yes No IDK — L is the intersection of two regular languages, and L is undecidable.

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

Yes No IDK — If L is not regular, then L is NP-hard.

Yes No IDK — If L is undecidable, then L is not regular.

Yes No IDK — If L is context-free, then L is infinite.

Yes No IDK — L is undecidable if and only if its reversal $L^R = \{w^R \mid w \in L\}$ is undecidable. (Recall that w^R denotes the reversal of the string w.)

Yes No IDK — L is undecidable if and only if its complement $\overline{L}$ is decidable.

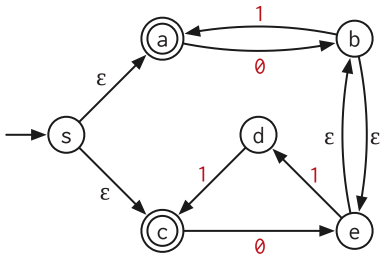
(c) Consider the following sets of undirected graphs:

- Trees is the set of all connected undirected graphs with no cycles.
- MostlyIndependent is the set of all undirected graphs that have an independent set containing at least half of the vertices. (Deciding whether a graph has this property is NP-hard.)

(For concreteness, assume that in both of these languages, graphs are represented by their adjacency matrices.) Which of the following ***must*** be true, assuming P≠NP?

Yes	No	IDK	Trees $\notin$ P
Yes	No	IDK	Trees $\subseteq$ MostlyIndependent
Yes	No	IDK	There is a polynomial-time reduction from Trees to MostlyIndependent
Yes	No	IDK	There is a polynomial-time reduction from MostlyIndependent to Trees
Yes	No	IDK	Trees is NP-hard.

(d) Let M be the following NFA:



Which of the following statements about M are true?

Yes	No	IDK	M rejects the empty string ε.
Yes	No	IDK	$\delta^*(s, \texttt{0101}) = \{a, d\}$
Yes	No	IDK	ε-reach$(s) = \{s, a, c\}$
Yes	No	IDK	M accepts the string $\texttt{01101011}$
Yes	No	IDK	$L(M) = (\texttt{011})^* + (\texttt{01})^*$

(e) Which of the following languages over the alphabet $\Sigma = \{0, 1\}$ are ***regular***? Recall that $\#(a, w)$ denotes the number of times symbol a appears in string w.

Yes No IDK $\left\{w \in \Sigma^* \;\middle|\; \#(1, w) \text{ is a perfect square}\right\}$

Yes No IDK The language generated by the context-free grammar $S \to 0S \mid S1 \mid \varepsilon$

Yes No IDK $\{w \in \Sigma^* \mid \#(0, w) + \#(1, w) < 374\}$

Yes No IDK $\{w \in \Sigma^* \mid \#(0, w) - \#(1, w) < 374\}$

Yes No IDK The complement of a regular language

(f) Which of the following languages or problems are ***decidable***?

Yes No IDK 3Color

Yes No IDK $\{\langle M \rangle \mid M$ accepts all non-empty strings but rejects the empty string $\varepsilon\}$

Yes No IDK $\{\langle M \rangle \mid M$ accepts every string whose length is a perfect square$\}$

Yes No IDK $\{\langle M \rangle \mid M$ is a Turing machine with at least two states$\}$

Yes No IDK $\varnothing$

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

Yes No IDK 3Color

Yes No IDK $\{\langle M \rangle \mid M$ accepts all non-empty strings but rejects the empty string $\varepsilon\}$

Yes No IDK $\{\langle M \rangle \mid M$ accepts every string whose length is a perfect square$\}$

Yes No IDK $\{\langle M \rangle \mid M$ is a Turing machine with at least two states$\}$

Yes No IDK $\varnothing$

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

$$\text{Marvin} := \big\{\langle M\rangle \bigm| M \text{ rejects an infinite number of strings}\big\}$$

Professor Beeblebrox, your instructor in Infinitely Improbable Galactic Presidencies, suggests a reduction from the standard halting language

$$\text{Halt} := \big\{(\langle M\rangle, w) \bigm| M \text{ halts on inputs } w\big\}.$$

Specifically, suppose there is a program ParanoidAndroid that decides Marvin. Professor Beeblebrox claims that the following algorithm decides Halt.

```
DecideHalt($(\langle M\rangle, w)$):
    Write code for the following algorithm:
        HeartOfGold($x$):
            run $M$ on input $w$
            if $x$ = VOGONPOETRY
                return False
            else
                return True
    return ParanoidAndroid($\langle$HeartOfGold$\rangle$)
```

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

Yes	No	IDK	If M accepts w, then HeartOfGold accepts VOGONPOETRY.
Yes	No	IDK	If M rejects w, then HeartOfGold rejects VOGONPOETRY.
Yes	No	IDK	If M hangs on w, then HeartOfGold rejects EDDIE.
Yes	No	IDK	ParanoidAndroid rejects $\langle$HeartOfGold$\rangle$.
Yes	No	IDK	DecideHalt decides Halt; that is, Professor Beeblebrox's proof is correct.

CS/ECE 374 A ✧ Fall 2019 Final Exam Problem 2	Gradescope name:

You and your friends are planning a hiking trip in Jellystone National Park over winter break. You have a map of the park's trails that lists all the scenic views in the park but also warns that certain trail segments have a high risk of bear encounters. To make the hike worthwhile, you want to see at least three scenic views. You also don't want to get eaten by a bear, so you are willing to hike at most one high-bear-risk segment. Because the trails are narrow, each trail segment allows traffic in only one direction.

Your friend has converted the map into a directed graph $G = (V, E)$, where V is the set of intersections and E is the set of trail segments. A subset S of the edges are marked as *Scenic*; another subset B of the edges are marked as *high-Bear-risk*. You may assume that $S \cap B = \varnothing$. Each segment $e \in E$ is also labeled with a positive length $\ell(e)$ in miles. Your campsite appears on the map as a particular vertex $s \in V$, and the visitor center is another vertex $t \in V$.

Describe and analyze an algorithm to compute the shortest hike from your campsite s to the visitor center t that includes *at least* three scenic views and *at most* one high-bear-risk trail segment. You may assume such a hike exists.

CS/ECE 374 A ✧ Fall 2019 **Final Exam Problem 3**	Gradescope name:

For each of the following languages over the alphabet {0, 1}, state whether the language is regular or not, and then justify your answer as follows:

- If the language is regular, ***either*** give an regular expression that describes the language, ***or*** draw/describe a DFA or NFA that accepts the language. You do not need to prove that your automaton or regular expression is correct.

- If the language is not regular, ***prove*** that the language is not regular.

[Hint: Exactly one of these languages is regular.]

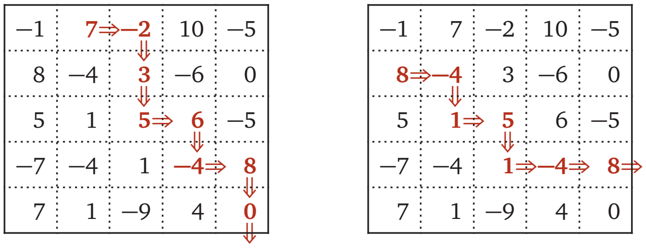
(a) $\{xy \mid x \text{ is a palindrome and } y \text{ is a palindrome}\}$

(b) $\{xy \mid x \text{ is a palindrome and } |x| \geq 2\}$

CS/ECE 374 A ✧ Fall 2019 **Final Exam Problem 4**	Gradescope name:

Vankin's Mile is an American solitaire game played on an $n \times n$ square grid. The player starts by placing a token on any square of the grid. Then on each turn, the player moves the token either one square to the right or one square down. The game ends when player moves the token off the edge of the board. Each square of the grid has a numerical value, which could be positive, negative, or zero. The player starts with a score of zero; whenever the token lands on a square, the player adds its value to his score. The object of the game is to score as many points as possible.

For example, given the grid shown below, the player can score $7-2+3+5+6-4+8+0 = 23$ points by following the path on the left, or they can score $8-4+1+5+1-4+8 = 15$ points by following the path on the right.



−1	**7** ⇒	**−2** ⇓	10	−5
8	−4	**3** ⇓	−6	0
5	1	**5** ⇒	**6** ⇓	−5
−7	−4	1	**−4** ⇒	**8** ⇓
7	1	−9	4	**0** ⇓

−1	7	−2	10	−5
8 ⇒	**−4** ⇓	3	−6	0
5	**1** ⇒	**5** ⇓	6	−5
−7	−4	**1** ⇒	**−4** ⇒	**8** ⇒
7	1	−9	4	0

(a) Describe and analyze an efficient algorithm to compute the maximum possible score for a game of Vankin's Mile, given the $n \times n$ array of values as input.

(b) A variant called *Vankin's Niknav* adds an additional constraint to Vankin's Mile: *The sequence of values that the token touches must be a* ***palindrome***. Thus, the example path on the right is valid, but the example path on the left is not. Describe and analyze an efficient algorithm to compute the maximum possible score for an instance of Vankin's Niknav, given the $n \times n$ array of values as input.

CS/ECE 374 A ✧ Fall 2019 **Final Exam Problem 5**	Gradescope name:

Recall that a *satisfying assignment* for a 3CNF Boolean formula Φ assigns values (True or False) to the variables of Φ so that Φ evaluates to True. A satisfying assignment is *balanced* if *exactly* half of the variables are set to True.

The Balanced3Sat problem asks whether a given 3CNF formula Φ has a balanced satisfying assignment. ***Prove*** that Balanced3Sat is NP-hard.

CS/ECE 374 A ✧ Fall 2019 Final Exam Problem 6	Gradescope name:

Let M be an arbitrary NFA *without* ε-transitions, with input alphabet $\Sigma = \{0, 1\}$. Describe and analyze an efficient algorithm to decide whether M accepts an infinite number of strings.

(scratch paper)

(scratch paper)

(scratch paper)

Some useful NP-hard problems. You are welcome to use any of these in your own NP-hardness proofs, except of course for the specific problem you are trying to prove NP-hard.

CircuitSat: Given a boolean circuit, are there any input values that make the circuit output True?

3Sat: Given a boolean formula in conjunctive normal form, with exactly three distinct literals per clause, does the formula have a satisfying assignment?

MaxIndependentSet: Given an undirected graph G, what is the size of the largest subset of vertices in G that have no edges among them?

MaxClique: Given an undirected graph G, what is the size of the largest complete subgraph of G?

MinVertexCover: Given an undirected graph G, what is the size of the smallest subset of vertices that touch every edge in G?

MinSetCover: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subcollection whose union is S?

MinHittingSet: Given a collection of subsets $S_1, S_2, \ldots, S_m$ of a set S, what is the size of the smallest subset of S that intersects every subset S_i?

3Color: Given an undirected graph G, can its vertices be colored with three colors, so that every edge touches vertices with two different colors?

HamiltonianPath: Given graph G (either directed or undirected), is there a path in G that visits every vertex exactly once?

HamiltonianCycle: Given a graph G (either directed or undirected), is there a cycle in G that visits every vertex exactly once?

TravelingSalesman: Given a graph G (either directed or undirected) with weighted edges, what is the minimum total weight of any Hamiltonian path/cycle in G?

LongestPath: Given a graph G (either directed or undirected, possibly with weighted edges), what is the length of the longest simple path in G?

SteinerTree: Given an undirected graph G with some of the vertices marked, what is the minimum number of edges in a subtree of G that contains every marked vertex?

SubsetSum: Given a set X of positive integers and an integer k, does X have a subset whose elements sum to k?

Partition: Given a set X of positive integers, can X be partitioned into two subsets with the same sum?

3Partition: Given a set X of $3n$ positive integers, can X be partitioned into n three-element subsets, all with the same sum?

IntegerLinearProgramming: Given a matrix $A \in \mathbb{Z}^{n \times d}$ and two vectors $b \in \mathbb{Z}^n$ and $c \in Z^d$, compute $\max\{c \cdot x \mid Ax \leq b, x \geq 0, x \in \mathbb{Z}^d\}$.

FeasibleILP: Given a matrix $A \in \mathbb{Z}^{n \times d}$ and a vector $b \in \mathbb{Z}^n$, determine whether the set of feasible integer points $\max\{x \in \mathbb{Z}^d \mid Ax \leq b, x \geq 0\}$ is empty.

Draughts: Given an $n \times n$ international draughts configuration, what is the largest number of pieces that can (and therefore must) be captured in a single move?

SuperMarioBrothers: Given an $n \times n$ Super Mario Brothers level, can Mario reach the castle?

SteamedHams: Aurora borealis? At this time of year, at this time of day, in this part of the country, localized entirely within your kitchen? May I see it?