CS/ECE 374 A ✧ Fall 2021

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 handout. 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⟩⟩*** or ***⟨⟨F19⟩⟩***, for example. Questions from this semester's homework (either written or on PrairieLearn) 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"—or worse, to *remember*—the solution to any particular problem, but to become more comfortable with solving a certain *type* of problem on your own. **"Understanding" is a seductive 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. ⟨⟨*Lab*⟩⟩

(b) Prove that $(w^R)^R = w$ for every string w. ⟨⟨*Lab*⟩⟩

(c) Prove that $|w| = |w^R|$ for every string w. ⟨⟨*Lab*⟩⟩

1.2 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 $\#(a, w)$.⟨⟨*Lab*⟩⟩

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

(c) Prove that $\#(a, w^R) = \#(a, w)$ for all symbols a and all strings w, where w^r denotes the reversal of w. ⟨⟨*Lab*⟩⟩

1.3 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 $n \geq 0$.

1.4 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: ⟨⟨***HW***⟩⟩

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

(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*** $\boldsymbol{w^c}$ 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:

$$shuffle(w,z) := \begin{cases} z & \text{if } w = \varepsilon \\ a \cdot shuffle(z,x) & \text{if } w = ax \end{cases}$$

For example, $shuffle(\texttt{0011},\texttt{0101}) = \texttt{00011011}$.

(a) Prove that $|shuffle(x,y)| = |x| + |y|$ for all strings x and y.

(b) Prove that $shuffle(w,w) = stutter(w)$ for every string w.

(c) Prove that $shuffle(odds(w), evens(w)) = w$ for every string w. ⟨⟨*HW*⟩⟩

(d) Prove that $evens(shuffle(w,z)) = z$ for all strings w and z such that $|w| = |z|$. ⟨⟨*HW*⟩⟩

Regular expressions

For each of the following languages over the alphabet $\Sigma = \{0, 1\}$, 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 start with the prefix `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 All strings containing either the substring `10` or the substring `01`, but not both. ***⟨⟨F16⟩⟩***

2.11 All strings that do not contain either `001` or `110` as a substring. ***⟨⟨F19⟩⟩***

2.12 All strings containing the subsequence `10` or the subsequence `01` (or possibly both).

2.13 All strings containing the subsequence `10` or the subsequence `01`, but not both.

2.14 All strings containing at least two `1`s and at least one `0`. ***⟨⟨Lab⟩⟩***

2.15 All strings containing at least two `1`s or at least one `0` (or possibly both).

2.16 All strings containing at least two `1`s or at least one `0`, but not both.

2.17 All strings in which every run of consecutive `0`s has even length. ***⟨⟨S21⟩⟩***

2.18 All strings in which every run of consecutive `0`s has even length and every run of consecutive `1`s has odd length. ***⟨⟨F14⟩⟩***

2.19 All strings whose length is divisible by 3.

2.20 All strings in which the number of `1`s is divisible by 3.

2.21 All strings in 0^*1^* whose length is divisible by 3. ***⟨⟨S14⟩⟩***

2.22 All strings in 0^*10^* whose length is divisible by 3. ***⟨⟨S18⟩⟩***

2.23 All strings in $0^*1^*0^*$ whose length is even. ***⟨⟨S18⟩⟩***

2.24 $\{0^n w 1^n \mid n > 1 \text{ and } q \in \Sigma^*\}$ ***⟨⟨S18⟩⟩***

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. Unless otherwise specified, all languages are over the alphabet $\Sigma = \{0, 1\}$.

2.1 The language {LONG, LUG, LEGO, LEG, LUG, LOG, LINGO} .

2.2 The language $\texttt{MOO}^* + \texttt{MEOO}^*\texttt{W}$

2.3 Every string of length at most 3.

2.4 All strings except 010.

2.5 All strings that end with the suffix 010.

2.6 All strings that do not start with the prefix 010.

2.7 All strings that contain the substring 010.

2.8 All strings that do not contain the substring 010.

2.9 All strings that contain the subsequence 010.

2.10 All strings containing the substring 10 or the substring 01.

2.11 All strings containing either the substring 10 or the substring 01, but not both. *⟨⟨F16⟩⟩*

2.12 All strings that do not contain either 001 or 110 as a substring. *⟨⟨F19⟩⟩*

2.13 All strings containing the subsequence 10 or the subsequence 01 (or possibly both).

2.14 All strings containing at least two 1s and at least one 0. *⟨⟨Lab⟩⟩*

2.15 All strings containing at least two 1s or at least one 0, but not both.

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

2.17 All strings in which every run of consecutive 0s has even length. *⟨⟨S21⟩⟩*

2.18 All strings in which every run of consecutive 0s has even length and every run of consecutive 1s has odd length. *⟨⟨F14⟩⟩*

2.19 All strings that end with 01 and that have odd length *⟨⟨S21⟩⟩*

2.20 All strings in which the number of 1s is divisible by 3.

2.21 All strings that represent an integer divisible by 3 in binary.

2.22 All strings that represent an integer divisible by 5 in base 7.

2.23 All strings in 0^*1^* whose length is divisible by 3. *⟨⟨S14⟩⟩*

2.24 All strings in 0^*10^* whose length is divisible by 3. *⟨⟨S18⟩⟩*

2.25 All strings in $0^*1^*0^*$ whose length is even. *⟨⟨S18⟩⟩*

2.26 $\{0^n w 1^n \mid n > 1 \text{ and } q \in \Sigma^*\}$ *⟨⟨S18⟩⟩*

Fooling sets

Prove that each of the following languages is *not* regular. Unless specified otherwise, all languages are over the alphabet $\Sigma = \{\texttt{0}, \texttt{1}\}$.

4.1 All strings with more `0`s than `1`s. ⟨⟨*S14*⟩⟩

4.2 All strings with fewer `0`s than `1`s.

4.3 All strings with exactly twice as many `0`s as `1`s. ⟨⟨*Lab*⟩⟩

4.4 All strings with at least twice as many `0`s as `1`s.

4.5 $\left\{\texttt{0}^{2^n} \mid n \geq 0\right\}$ ⟨⟨*Lab*⟩⟩

4.6 $\left\{\texttt{0}^{3^n} \mid n \geq 0\right\}$ ⟨⟨*S21*⟩⟩

4.7 $\left\{\texttt{0}^{F_n} \mid n \geq 0\right\}$, 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 $\min\{i, j\} \leq 2$.]

4.8 $\left\{\texttt{0}^{n^2} \mid n \geq 0\right\}$ ⟨⟨*Lab*⟩⟩

4.9 $\left\{\texttt{0}^{n^3} \mid n \geq 0\right\}$

4.10 $\left\{\texttt{0}^{2n}\texttt{1}^n \mid n \geq 0\right\}$ ⟨⟨*Lab*⟩⟩

4.11 $\{\texttt{0}^m\texttt{1}^n \mid m \neq 2n\}$ ⟨⟨*Lab*⟩⟩

4.12 $\left\{\texttt{0}^i\texttt{1}^j\texttt{0}^k \mid 2i = k \text{ or } i = 2k\right\}$ ⟨⟨*S18*⟩⟩

4.13 $\left\{\texttt{0}^i\texttt{1}^j\texttt{0}^k \mid i + j = 2k\right\}$ ⟨⟨*F19*⟩⟩

4.14 $\left\{x\texttt{\#}y \mid x, y \in \{\texttt{0}, \texttt{1}\}^* \text{ and } \#(\texttt{0}, x) = \#(\texttt{1}, y)\right\}$

4.15 $\left\{xx^c \mid x \in \{\texttt{0}, \texttt{1}\}^*\right\}$, where x^c is the *complement* of x, obtained by replacing every `0` in x with a `1` and vice versa. For example, $\texttt{0001101}^c = \texttt{1110010}$.

4.16 Properly balanced strings of parentheses, described by the context-free grammar $S \to \varepsilon \mid SS \mid \texttt{(}S\texttt{)}$. ⟨⟨*Lab*⟩⟩

4.17 Palindromes whose length is divisible by 3.

4.18 Strings in which at least two runs of consecutive `0`s have the same length.

4.19 $\left\{(\texttt{01})^n(\texttt{10})^n \mid n \geq 0\right\}$

4.20 $\left\{(\texttt{01})^m(\texttt{10})^n \mid n \geq m \geq 0\right\}$

4.21 $\left\{w\texttt{\#}x\texttt{\#}y \mid w, x, y \in \Sigma^* \text{ and } w, x, y \text{ are not all equal}\right\}$

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 $\Sigma = \{\mathtt{0}, \mathtt{1}\}$. Read the language descriptions ***very*** carefully.

5.1 The set of all strings in $\{\mathtt{0}, \mathtt{1}\}^*$ 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`.) ⟨⟨***F14***⟩⟩

5.2 The set of all strings in $\{\mathtt{0}, \mathtt{1}\}^*$ 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`.) ⟨⟨***F14***⟩⟩

5.3 $\{www \mid w \in \Sigma^*\}$ ⟨⟨***F14***⟩⟩

5.4 $\{wxw \mid w, x \in \Sigma^*\}$ ⟨⟨***F14***⟩⟩

5.5 All strings such that in every prefix, the number of `0`s is greater than the number of `1`s.

5.6 All strings such that in every *non-empty* prefix, the number of `0`s is greater than the number of `1`s.

5.7 $\{\mathtt{0}^m\mathtt{1}^n \mid 0 \le m - n \le 374\}$

5.8 $\{\mathtt{0}^m\mathtt{1}^n \mid 0 \le m + n \le 374\}$

5.9 The language generated by the following context-free grammar:

$$S \to \mathtt{0}A\mathtt{1} \mid \varepsilon$$
$$A \to \mathtt{1}S\mathtt{0} \mid \varepsilon$$

5.10 The language generated by the following free grammar $S \to \mathtt{0}S\mathtt{1} \mid \mathtt{1}S\mathtt{0} \mid \varepsilon$

5.11 $\{w\#x \mid w, x \in \{\mathtt{0}, \mathtt{1}\}^* \text{ and no substring of } w \text{ is also a substring of } x\}$

5.12 $\{w\#x \mid w, x \in \{\mathtt{0}, \mathtt{1}\}^* \text{ and no \textit{non-empty} substring of } w \text{ is also a substring of } x\}$

5.13 $\{w\#x \mid w, x \in \{\mathtt{0}, \mathtt{1}\}^* \text{ and \textit{every} non-empty substring of } w \text{ is also a substring of } x\}$

5.14 $\{w\#x \mid w, x \in \{\mathtt{0}, \mathtt{1}\}^* \text{ and } w \text{ is a substring of } x\}$

5.15 $\{w\#x \mid w, x \in \{\mathtt{0}, \mathtt{1}\}^* \text{ and } w \text{ is a proper substring of } x\}$

5.16 $\{xy \mid x \text{ is a palindrome and } y \text{ is a palindrome}\}$ ⟨⟨***F19***⟩⟩

5.17 $\{xy \mid x \text{ is not a palindrome}\}$ ⟨⟨***F19***⟩⟩

5.18 $\{xy \mid x \text{ is a palindrome and } |x| > 1\}$ ⟨⟨***F19***⟩⟩

5.19 $\{xy \mid \#(\mathtt{0}, x) = \#(\mathtt{1}, y) \text{ and } \#(\mathtt{1}, x) = \#(\mathtt{0}, y)\}$

5.20 $\{xy \mid \#(\mathtt{0}, x) = \#(\mathtt{1}, y) \text{ or } \#(\mathtt{1}, x) = \#(\mathtt{0}, y)\}$

Product/Subset Constructions

For each of the following languages L over the alphabet $\{0, 1\}$, formally describe a DFA $M = (Q, s, A, \delta)$ that recognizes L. ***Do not attempt to draw the DFA. Do not use the phrase "product construction".*** 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.

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 5.

  (c) The length $|w|$ is divisible by 7.

  (d) The binary value of w is divisible by 7.

  (e) w represents a number divisible by 5 in base 7.

  (f) w contains the substring 00

  (g) w does not contain the substring 11

  (h) 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 is regular.

7.1 All strings in L whose length is divisible by 3.

7.2 $\textsc{OneInFront}(L) := \{\mathtt{1}x \mid x \in L\}$

7.3 $\textsc{OnlyOnes}(L) := \left\{\mathtt{1}^{\#(\mathtt{1},w)} \;\middle|\; w \in L\right\}$

7.4 $\textsc{OnlyOnes}^{-1}(L) := \left\{w \;\middle|\; \mathtt{1}^{\#(\mathtt{1},w)} \in L\right\}$

7.5 $\textsc{MissingFirstOne}(L) := \{w \in \Sigma^* \mid \mathtt{1}w \in L\}$

7.6 $\textsc{MissingOneOne}(L) := \{xy \mid x\mathtt{1}y \in L\}$

7.7 $\textsc{Prefixes}(L) := \{x \mid xy \in L \text{ for some } y \in \Sigma^*\}$

7.8 $\textsc{Suffixes}(L) := \{y \mid xy \in L \text{ for some } x \in \Sigma^*\}$ *⟨⟨F16⟩⟩*

7.9 $\textsc{Evens}(L) := \{\mathit{evens}(w) \mid w \in L\}$, 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 $\mathit{odds}(\mathtt{0001101}) = \mathtt{0011}$. *⟨⟨F14⟩⟩*

7.10 $\textsc{Evens}^{-1}(L) := \{w \mid \mathit{evens}(w) \in L\}$, where the functions *evens* and *odds* are recursively defined as above. *⟨⟨F14⟩⟩*

7.11 $\textsc{AddParity}(L) = \{\mathit{addparity}(w) \mid w \in L\}$, where *⟨⟨S18⟩⟩*

$$\mathit{addparity}(w) = \begin{cases} \mathtt{0}w & \text{if } \#(\mathtt{1}, w) \text{ is even} \\ \mathtt{1}w & \text{if } \#(\mathtt{1}, w) \text{ is odd} \end{cases}$$

7.12 $\textsc{StripFinal}\mathtt{0}\textsc{s}(L) = \{w \mid w\mathtt{0}^n \in L \text{ for some } n \ge 0\}$. Less formally, $\textsc{StripFinal}\mathtt{0}\textsc{s}(L)$ is the set of all strings obtained by removing any number of final $\mathtt{0}$s from strings in L. *⟨⟨S18⟩⟩*

7.13 $\textsc{Obliviate}(L) := \{\mathit{obliviate}(w) \mid w \in L\}$, where $\mathit{obliviate}(w)$ is the string obtained from w by deleting every $\mathtt{1}$. *⟨⟨F19⟩⟩*

7.14 $\textsc{UnObliviate}(L) := \{w \in \Sigma^* \mid \mathit{obliviate}(w) \in L\}$, where $\mathit{obliviate}(w)$ is the string obtained from w by deleting every $\mathtt{1}$. *⟨⟨F19⟩⟩*

7.15 $\textsc{SameSlash}(w) = \{\mathit{sameslash}(w) \mid w \in L\}$, where $\mathit{sameslash}(w)$ is 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. *⟨⟨F19⟩⟩*

7.16 $\textsc{DiffSlash}(w) = \{\mathit{diffslash}(w) \mid w \in L\}$, where $\mathit{diffslash}(w)$ is the string in $\{\mathtt{0}, \mathtt{1}, \mathtt{/}\}$ obtained from w by inserting a new symbol $\mathtt{/}$ between any two consecutive symbols that are ***not*** equal. *⟨⟨F19⟩⟩*

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. Unless specified otherwise, all languages are over the alphabet $\{0, 1\}$. 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 $\{0^n 1^{2n} \mid n \geq 0\}$

8.4 $\{0^m 1^n \mid n \neq 2m\}$

8.5 $\{0^i 1^j 0^{i+j} \mid i, j \geq 0\}$

8.6 $\{0^{i+j} \# 0^j \# 0^i \mid i, j \geq 0\}$

8.7 $\{0^i 1^j 2^k \mid j \neq i + k\}$

8.8 $\{0^i 1^j 2^k \mid i = 2k \text{ or } 2i = k\}$ *⟨⟨S18⟩⟩*

8.9 $\{0^i 1^j 2^k \mid i + j = 2k\}$ *⟨⟨F19⟩⟩*

8.10 $\left\{w \# 0^{\#(0,w)} \,\middle|\, w \in \{0, 1\}^*\right\}$

8.11 $\{0^i 1^j 2^k \mid i = j \text{ or } j = k \text{ or } i = k\}$

8.12 $\{0^i 1^j 2^k \mid i \neq j \text{ or } j \neq k\}$

8.13 $\{0^{2i} 1^{i+j} 2^{2j} \mid i, j \geq 0\}$

8.14 $\left\{x \# y^R \,\middle|\, x, y \in \{0, 1\}^* \text{ and } x \neq y\right\}$

8.15 All strings in $\{0, 1\}^*$ that are *not* palindromes.

8.16 $\left\{0^n 1^{an+b} \,\middle|\, n \geq 0\right\}$, where a and b are arbitrary fixed natural numbers.

8.17 $\left\{0^n 1^{an-b} \,\middle|\, n \geq b/a\right\}$, where a and b are arbitrary fixed natural numbers.

True or False (sanity check)

For each statement below, check "Yes" if the statement is ***ALWAYS*** true and "No" otherwise, and give a ***brief*** explanation of your answer. For example:

[✓ Yes] No — If $2+2=5$ then Jeff is the Queen of England.
The hypothesis is false, so the implication is true.

Yes [✓ No] — $x+y$ is even.
Suppose $x=1$ and $y=0$.

[✓ Yes] No — The set of all binary strings with an even number of 1s is regular.
`0*(10*10*)*`
—*or*—
Accepted by 2-state DFA, where current state $= \#1s \bmod 2$.

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.

A.4 For every language L, if L is regular then L can be represented by a regular expression.

A.5 For every language L, if L is not regular then L cannot be represented by a regular expression.

A.6 For every language L, if L can be represented by a regular expression, then L is regular.

A.7 For every language L, if L cannot be represented by a regular expression, then L is not regular.

A.8 For every language L, if there is a DFA that accepts every string in L, then L is regular.

A.9 For every language L, if there is a DFA that accepts every string not in L, then L is not regular.

A.10 For every language L, if there is a DFA that rejects every string not in L, then L is regular.

A.11 For every language L, if for every string $w \in L$ there is a DFA that accepts w, then L is regular.
⟨⟨*S14*⟩⟩

A.12 For every language L, if for every string $w \notin L$ there is a DFA that rejects w, then L is regular.

A.13 For every language L, if some DFA recognizes L, then some NFA also recognizes L.

A.14 For every language L, if some NFA recognizes L, then some DFA also recognizes L.

A.15 For every language L, if some NFA with ε-transitions recognizes L, then some NFA without ε-transitions also recognizes L.

A.16 For every language L, and for every string $w \in L$, there is a DFA that accepts w. ⟪*F19*⟫

A.17 Every regular language is recognized by a DFA with exactly 374 accepting states. ⟪*F19*⟫

A.18 Every regular language is recognized by an NFA with exactly 374 accepting states. ⟪*F19*⟫

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 every language L, the language L^* is regular. ⟪*F14, F16*⟫

B.12 For every language 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 every language 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 For all languages $L \subseteq \Sigma^*$, if L contains all but a finite number of strings of Σ^*, then L is regular. ⟪*S14*⟫

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 not regular.

B.32 If L is not regular and $L \cap L' = \varnothing$, then L' is regular. ⟨⟨*F16*⟩⟩

B.33 If L is regular and L' is not regular, then $L \cap L' = \varnothing$.

B.34 If $L \subseteq L'$ and L is regular, then L' is regular.

B.35 If $L \subseteq L'$ and L' is regular, then L is regular. ⟨⟨*F14*⟩⟩

B.36 If $L \subseteq L'$ and L is not regular, then L' is not regular.

B.37 If $L \subseteq L'$ and L' is not regular, then L is not regular. ⟨⟨*F14*⟩⟩

B.38 Two languages L and L' are regular if and only if $L \cap L'$ is regular. ⟨⟨*F19*⟩⟩

B.39 For all languages $L \subseteq \Sigma^*$, if L cannot be described by a regular expression, then some DFA accepts $\Sigma^* \setminus L$.

B.40 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.41 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.42 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 For all languages $L \subseteq \Sigma^*$, if L cannot be recognized by a DFA, then L is context-free.

C.2 For all languages $L \subseteq \Sigma^*$, if L cannot be recognized by a DFA, then L is not context-free.

C.3 For all languages $L \subseteq \Sigma^*$, if L can be recognized by a DFA, then L is context-free.

C.4 For all languages $L \subseteq \Sigma^*$, if L can be recognized by a DFA, then L is not context-free.

C.5 For all languages $L \subseteq \Sigma^*$, if L is not context-free, then L is regular.

C.6 For all languages $L \subseteq \Sigma^*$, if L is not context-free, then $\Sigma^* \setminus L$ is regular.

C.7 For all languages $L \subseteq \Sigma^*$, if L is not context-free, then L is not regular.

C.8 For all languages $L \subseteq \Sigma^*$, if L is not context-free, then $\Sigma^* \setminus L$ is not regular.

C.9 The empty language is context-free. ⟨⟨*F19*⟩⟩

C.10 Every finite language is context-free.

C.11 Every context-free language is regular. ⟨⟨*F14*⟩⟩

C.12 Every regular language is context-free.

C.13 Every non-context-free language is non-regular. ⟨⟨*F16*⟩⟩

C.14 Every language is either regular or context-free. ⟨⟨*F19*⟩⟩

C.15 For all context-free languages L and L', the language $L \bullet L'$ is also context-free. ⟨⟨*F16*⟩⟩

C.16 For every context-free language L, the language L^* is also context-free.

C.17 For all context-free languages A, B, and C, the language $(A \cup B)^* \bullet C$ is also context-free.

C.18 For every language L, the language L^* is context-free.

C.19 For every language L, if L^* is context-free then L is context-free.

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 every language L, if L is regular, then $\equiv_L$ has finitely many equivalence classes.

D.2 For every language L, if L is not regular, then $\equiv_L$ has infinitely many equivalence classes. ⟨⟨*S14*⟩⟩

D.3 For every language L, if $\equiv_L$ has finitely many equivalence classes, then L is regular.

D.4 For every language 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 every language L, each equivalence class of $\equiv_L$ is a regular language.

Fooling Sets

E.1 If a language L has an infinite fooling set, then L is not regular.

E.2 If a language L has an finite fooling set, then L is regular.

E.3 If a language L does not have an infinite fooling set, then L is regular.

E.4 If a language L is not regular, then L has an infinite fooling set.

E.5 If a language L is regular, then L has no infinite fooling set.

E.6 If a language L is not regular, then L has no finite fooling set. *⟨⟨F14, F16⟩⟩*

E.7 If a language L has a fooling set of size 374, then L is not regular. *⟨⟨F19⟩⟩*

E.8 If a language L does not have a fooling set of size 374, then L is regular. *⟨⟨F19⟩⟩*

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 $\{0^i 1^j 2^k \mid i + j - k = 374\}$ is regular. *⟨⟨S14⟩⟩*

F.2 $\{0^i 1^j 2^k \mid i + j - k \leq 374\}$ is regular.

F.3 $\{0^i 1^j 2^k \mid i + j + k = 374\}$ is regular.

F.4 $\{0^i 1^j 2^k \mid i + j + k > 374\}$ is regular.

F.5 $\{0^i 1^j \mid i < 374 < j\}$ is regular. *⟨⟨S14⟩⟩*

F.6 $\left\{0^m 1^n \;\middle|\; 0 \leq m + n \leq 374\right\}$ is regular. *⟨⟨F14⟩⟩*

F.7 $\left\{0^m 1^n \;\middle|\; 0 \leq m - n \leq 374\right\}$ is regular. *⟨⟨F14⟩⟩*

F.8 $\{0^i 1^j \mid i, j \geq 0\}$ is not 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 $\left\{0^{374n} \;\middle|\; n \geq 0\right\}$ is regular. *⟨⟨F19⟩⟩*

F.16 $\left\{0^{37n} 1^{4n} \;\middle|\; n \geq 374\right\}$ is regular. *⟨⟨F19⟩⟩*

F.17 $\left\{0^{37n} 1^{4n} \;\middle|\; n \leq 374\right\}$ is regular. *⟨⟨F19⟩⟩*

F.18 $\{w \in \{0,1\}^* \mid |w| \text{ is divisible by } 374\}$ is regular.

F.19 $\{w \in \{0,1\}^* \mid w \text{ represents a integer divisible by } 374 \text{ in binary}\}$ is regular.

F.20 $\{w \in \{0,1\}^* \mid w \text{ represents a integer divisible by } 374 \text{ in base } 473\}$ is regular.

F.21 $\left\{w \in \{0,1\}^* \;\middle|\; |\#(0,w) - \#(1,w)| < 374\right\}$ is regular.

F.22 $\left\{w \in \{0,1\}^* \;\middle|\; |\#(0,x) - \#(1,x)| < 374 \text{ for every prefix } x \text{ of } w\right\}$ is regular.

F.23 $\left\{w \in \{0,1\}^* \;\middle|\; |\#(0,x) - \#(1,x)| < 374 \text{ for every substring } x \text{ of } w\right\}$ is regular.

F.24 $\left\{w0^{\#(0,w)} \;\middle|\; w \in \{0,1\}^*\right\}$ is regular.

F.25 $\left\{w0^{\#(0,w) \bmod 374} \;\middle|\; w \in \{0,1\}^*\right\}$ is regular.

Playing with Automata

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 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.6 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.7 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.8 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 $\left\{w^R \mid w \in L\right\}$ is also regular.

H.2 For every language L, if the language $\left\{w^R \mid w \in L\right\}$ is regular, then L is also regular. *⟨⟨F14⟩⟩*

H.3 For every language L, if the language $\left\{w^R \mid w \in L\right\}$ is not regular, then L is also not regular. *⟨⟨F14⟩⟩*

H.4 For every regular language L, the language $\left\{w \mid ww^R \in L\right\}$ is also regular.

H.5 For every regular language L, the language $\left\{ww^R \mid w \in L\right\}$ is also regular.

H.6 For every language L, if the language $\left\{w \mid ww^R \in L\right\}$ is regular, then L is also regular. *[Hint: Consider the language $L = \{0^n 1^n \mid n \geq 0\}$.]*

H.7 For every regular language L, the language $\left\{0^{|w|} \mid w \in L\right\}$ is also regular.

H.8 For every language L, if the language $\left\{0^{|w|} \mid w \in L\right\}$ is regular, then L is also regular.

H.9 For every context-free language L, the language $\left\{w^R \mid w \in L\right\}$ is also context-free.

CS/ECE 374 A ✧ Fall 2021 Fake Midterm 1 Problem 1	Name:

For each statement below, check "Yes" if the statement is ***ALWAYS*** true and "No" otherwise, and give a ***brief*** explanation of your answer.

(a) Every integer in the empty set is prime.

Yes No

(b) The language $\{0^m 1^n \mid m + n \le 374\}$ is regular.

Yes No

(c) The language $\{0^m 1^n \mid m - n \le 374\}$ is regular.

Yes No

(d) For all languages L, the language L^* is regular.

Yes No

(e) For all languages L, the language L^* is infinite.

Yes No

(f) 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

(g) For all languages L and L', if $L \cap L' = \varnothing$ and L' is not regular, then L is regular.

Yes No

(h) Every regular language is recognized by a DFA with exactly one accepting state.

Yes No

(i) Every regular language is recognized by an NFA with exactly one accepting state.

Yes No

(j) Every language is either regular or context-free.

Yes No

CS/ECE 374 A ✧ Fall 2021 **Fake Midterm 1 Problem 2**	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. $\left\{ 0^n w 0^n \;\middle|\; w \in \Sigma^+ \text{ and } n > 0 \right\}$

2. $\left\{ w 0^n w \;\middle|\; w \in \Sigma^+ \text{ and } n > 0 \right\}$

CS/ECE 374 A ✧ Fall 2021 **Fake Midterm 1 Problem 3**	Name:

The *parity* of a bit-string w is 0 if w has an even number of 1s, and 1 if w has an odd number of 1s. 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 2021 **Fake Midterm 1 Problem 4**	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 2021 **Fake Midterm 1 Problem 5**	Name:

For any string $w \in \{0,1\}^*$, let $\mathit{obliviate}(w)$ denote the string obtained from w by removing every 1. For example:

$$\begin{aligned}\mathit{obliviate}(\varepsilon) &= \varepsilon\\ \mathit{obliviate}(000000) &= 000000\\ \mathit{obliviate}(111111) &= \varepsilon\\ \mathit{obliviate}(010001101) &= 00000\end{aligned}$$

Let L be an arbitrary regular language.

1. ***Prove*** that the language $\textsc{Obliviate}(L) = \{\mathit{obliviate}(w) \mid w \in L\}$ is regular.

2. ***Prove*** that the language $\textsc{Unobliviate}(L) = \{w \in \{0,1\}^* \mid \mathit{obliviate}(w) \in L\}$ is regular.

CS/ECE 374 A ✧ Fall 2021

Midterm 1

September 27, 2021

Directions

- ***Don't panic!***
- If you brought anything except your writing implements, your **hand-written** double-sided 8½" × 11" cheat sheet, please put it away for the duration of the exam. In particular, please turn off and put away *all* medically unnecessary electronic devices.
- The exam has five numbered questions.
- Write your answers on blank white paper. Please start your solution to each numbered question on a new sheet of paper.
- You have 150 minutes to write, scan, and submit your solutions. The exam is designed to take at most 120 minutes to complete. We are providing 30 minutes of slack to scan and submit in case of unforeseen technology issues.
- If you are ready to scan your solutions before 9:15pm, send a private message to the host ("Ready to scan") and wait for confirmation before leaving the Zoom call.
- Please scan ***all*** paper that you used during the exam — first your solutions, in the correct order, then your cheat sheet (if any), and finally any scratch paper.
- Proofs are required for full credit if and only if we explicitly ask for them, using the word ***prove*** in bold italics. In particular, if we ask you to show that a language is regular, you can provide a regular expression, DFA, NFA, or boolean combination *without justification*. Similarly, if we ask you to give a DFA or NFA, you to *not* have to name or describe the states.
- Finally, if something goes seriously wrong, send email to jeffe@illinois.edu as soon as possible explaining the situation. If you have already finished the exam but cannot submit to Gradescope for some reason, include a complete scan of your exam in your email. If you are in the middle of the exam, send Jeff email, finish the exam (if you can) within the time limit, and then send a second email with your completed exam.

---

1. 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 and proving that the set you construct is indeed a fooling set for that language).

   (a) $\{0^p 1^q 0^r \mid r = p + q\}$

   (b) $\{0^p 1^q 0^r \mid r = p + q \bmod 2\}$
   [Hint: First think about the language $\{0^p 1^q \mid q = p \bmod 2\}$*]*

2. Let L be any regular language over the alphabet $\Sigma = \{0, 1\}$.

   Let take2skip2(w) be a function takes an input string w and returns the subsequence of symbols at positions $1, 2, 5, 6, 9, 10, \ldots 4i+1, 4i+2, \ldots$ in w. In other words, take2skip2(w) takes the first two symbols of w, skip the next two, takes the next two, skips the next two, and so on. For example:

$$\text{take2skip2}(\underline{1}) = 1$$
$$\text{take2skip2}(\underline{01}0) = 01$$
$$\text{take2skip2}(\underline{01}00\underline{11}11\underline{00}01\underline{1}) = 0111001$$

   Choose exactly one of the following languages, and prove that your chosen language is regular. (In fact, *both* languages are regular, but we only want a proof for one of them.) Don't forget to tell us which language you've chosen!

   (a) $L_1 = \{w \in \Sigma^* \mid \text{take2skip2}(w) \in L\}$.

   (b) $L_2 = \{\text{take2skip2}(w) \mid w \in L\}$.

3. Prove that the following languages are not regular by building an infinite fooling set for each of them. For each language, prove that the set you constructed is indeed a fooling set.

   (a) $\{0^p 1^q 0^r \mid r > 0 \quad \text{and} \quad q \bmod r = 0 \quad \text{and} \quad p \bmod r = 0\}$

   (b) $\{0^p 1^q \mid q > 0 \quad \text{and} \quad p = q^q\}$

4. Consider the following recursive function:

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

   For example, MINGLE$(01, 10) = $ MINGLE$(1, 0100) = $ MINGLE$(\varepsilon, 101001) = 101001$.

   (a) Prove that $|\text{MINGLE}(w, z)| = 2|w| + |z|$ for all strings w and z.

   (b) Prove that $\text{MINGLE}(w, z \bullet z^R) = (\text{MINGLE}(w, z \bullet z^R))^R$ for all strings w and z.

(There's one more question on the next page)

5. For each statement below, write "Yes" if the statement is always true and write "No" otherwise, and give a brief (one short sentence) explanation of your answer. Read these statements very carefully—small details matter!

   (a) If L is a regular language over the alphabet $\{0,1\}$, then $\{w1w \mid w \in L\}$ is also regular.

   (b) If L is a regular language over the alphabet $\{0,1\}$, then $\{x1y \mid x,y \in L\}$ is also regular.

   (c) The context-free grammar $S \to 0S1 \mid 1S0 \mid SS \mid 01 \mid 10$ generates the language $(0+1)^+$

   (d) Every regular expression that does not contain a Kleene star (or Kleene plus) represents a finite language.

   (e) Let L_1 be a finite language and L_2 be an arbitrary language. Then $L_1 \cap L_2$ is regular.

   (f) Let L_1 be a finite language and L_2 be an arbitrary language. Then $L_1 \cup L_2$ is regular.

   (g) The regular expression $(00+01+10+11)^*$ represents the language of all strings over $\{0,1\}$ of even length.

   (h) The ε-reach of any state in an NFA contains the state itself.

   (i) The language $L = 0^*$ over the alphabet $\Sigma = \{0,1\}$ has a fooling set of size 2.

   (j) Suppose we define an ε-DFA to be a DFA that can additionally make ε-transitions. Any language that can be recognized by an ε-DFA can also be recognized by a DFA that does not make any ε-transitions.

CS/ECE 374 A ✧ Fall 2021

Conflict Midterm 1

September 28, 2021

Directions

- ***Don't panic!***
- If you brought anything except your writing implements, your **hand-written** double-sided 8½" × 11" cheat sheet, please put it away for the duration of the exam. In particular, please turn off and put away *all* medically unnecessary electronic devices.
- The exam has five numbered questions.
- Write your answers on blank white paper. Please start your solution to each numbered question on a new sheet of paper.
- You have 150 minutes to write, scan, and submit your solutions. The exam is designed to take at most 120 minutes to complete. We are providing 30 minutes of slack to scan and submit in case of unforeseen technology issues.
- If you are ready to scan your solutions before 9:15pm, send a private message to the host ("Ready to scan") and wait for confirmation before leaving the Zoom call.
- Please scan ***all*** paper that you used during the exam — first your solutions, in the correct order, then your cheat sheet (if any), and finally any scratch paper.
- Proofs are required for full credit if and only if we explicitly ask for them, using the word ***prove*** in bold italics. In particular, if we ask you to show that a language is regular, you can provide a regular expression, DFA, NFA, or boolean combination *without justification*. Similarly, if we ask you to give a DFA or NFA, you to *not* have to name or describe the states.
- Finally, if something goes seriously wrong, send email to jeffe@illinois.edu as soon as possible explaining the situation. If you have already finished the exam but cannot submit to Gradescope for some reason, include a complete scan of your exam in your email. If you are in the middle of the exam, send Jeff email, finish the exam (if you can) within the time limit, and then send a second email with your completed exam.

---

1. For each of the following languages over the alphabet $\Sigma = \{\texttt{0}, \texttt{1}\}$, 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 and proving that the set you construct is indeed a fooling set for that language).

   (a) $\{\texttt{0}^p\texttt{1}^q\texttt{0}^r \mid p = (q + r) \bmod 2\}$

   (b) $\{\texttt{0}^p\texttt{1}^q\texttt{0}^r \mid p = q + r\}$

2. Let L be any regular language over the alphabet $\Sigma = \{\texttt{0}, \texttt{1}\}$.

   Let $\mathsf{compress}(w)$ be a function that takes a string w as input, and returns the string formed by compressing every run of $\texttt{0}$s in w by half. Specifically, every run of $2n$ $\texttt{0}$s is compressed to length n, and every run of $2n + 1$ $\texttt{0}$s is compressed to length $n + 1$. For example:

$$\mathsf{compress}(\underline{\texttt{00000}}\texttt{11}\underline{\texttt{000}}\texttt{1}) = \underline{\texttt{000}}\texttt{11}\underline{\texttt{00}}\texttt{1}$$
$$\mathsf{compress}(\texttt{11}\underline{\texttt{0000}}\texttt{1}\underline{\texttt{0}}) = \texttt{11}\underline{\texttt{00}}\texttt{1}\underline{\texttt{0}}$$
$$\mathsf{compress}(\texttt{11111}) = \texttt{11111}$$

   Choose exactly one of the following languages, and prove that your chosen language is regular. (In fact, *both* languages are regular, but we only want a proof for one of them.) Don't forget to tell us which language you've chosen!

   (a) $\left\{w \in \Sigma^* \,\middle|\, \mathsf{compress}(w) \in L\right\}$

   (b) $\left\{\mathsf{compress}(w) \,\middle|\, w \in L\right\}$

3. Recall that the *greatest common divisor* of two positive integers p and q, written $\gcd(p, q)$, is the largest positive integer r that divides both p and q. For example, $\gcd(21, 15) = 3$ and $\gcd(3, 74) = 1$.

   Prove that the following languages are not regular by building an infinite fooling set for each of them. For each language, prove that the set you constructed is indeed a fooling set.

   (a) $\{\texttt{0}^p\texttt{1}^q\texttt{0}^r \mid p > 0 \text{ and } q > 0 \text{ and } r = \gcd(p, q)\}$.

   (b) $\{\texttt{0}^p\texttt{1}^{pq} \mid p > 0 \text{ and } q > 0\}$

4. Consider the following recursive function, RO (short for remove-ones) that operates on any string $w \in \Sigma^*$, where $\Sigma = \{0, 1\}$:

$$\mathsf{RO}(w) := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ \texttt{0} \cdot \mathsf{RO}(x) & \text{if } w = \texttt{0} \cdot x \text{ for some string } x \\ \mathsf{RO}(x) & \text{if } w = \texttt{1} \cdot x \text{ for some string } x \end{cases}$$

   (a) Prove that $|\mathsf{RO}(w)| \le |w|$ for all strings w.

   (b) Prove that $\mathsf{RO}(\mathsf{RO}(w)) = \mathsf{R0}(w)$ for all strings w.

(There's one more question on the next page)

5. For each statement below, write "Yes" if the statement is always true and write "No" otherwise, and give a brief (one short sentence) explanation of your answer. Read these statements very carefully—small details matter!

   (a) $\{0^n 1 \mid n > 0\}$ is the only infinite fooling set for the language $\{0^n 1 0^n \mid n > 0\}$.

   (b) $\{0^n 1 0^n \mid n > 0\}$ is a context-free language.

   (c) The context-free grammar $S \rightarrow 00S \mid S11 \mid 01$ generates the language $0^n 1^n$.

   (d) Any language that can be decided by an NFA with ε-transitions can also be decided by an NFA without ε-transitions.

   (e) For any string $w \in (0+1)^*$, let w^C denote the string obtained by flipping every 0 in w to 1, and every 1 in w to 0.
   If L is a regular language over the alphabet $\{0, 1\}$, then $\{ww^C \mid w \in L\}$ is also regular.

   (f) For any string $w \in (0+1)^*$, let w^C denote the string obtained by flipping every 0 in w to 1, and every 1 in w to 0.
   If L is a regular language over the alphabet $\{0, 1\}$, then $\{xy^C \mid x, y \in L\}$ is also regular.

   (g) The ε-reach of any state in an NFA contains the state itself.

   (h) Let L_1, L_2 be two regular languages. The language $(L_1 + L_2)^*$ is also regular.

   (i) The regular expression $(00 + 11)^*$ represents the language of all strings over $\{0, 1\}$ of even length.

   (j) The language $\{0^{2p} \mid p \text{ is prime}\}$ is regular.

CS/ECE 374 A ✧ Fall 2021
Midterm 2 Study Questions

This is a "core dump" of potential questions for Midterm 2. This should give you a good idea of the *types* of questions that we will ask on the exam, 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, for example, ***⟨⟨S18⟩⟩*** or ***⟨⟨F19⟩⟩***. 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)? Do you keep trying to modify the algorithm instead of modifying the graph?

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.

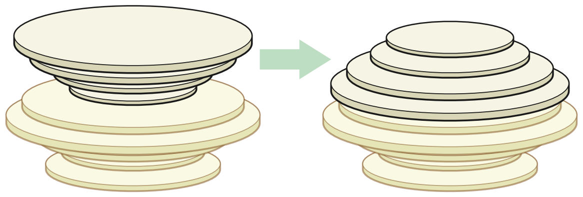


Figure 1. Flipping the top four pancakes.

   (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?

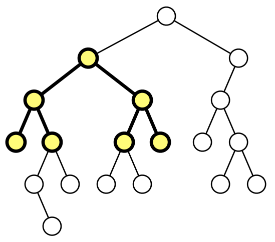
   *[Hint: This problem has **nothing** to do with the Tower of Hanoi!]*

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. ⟨⟨*F19*⟩⟩

   (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 `AREDIVIDER` 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`).

3. ⟨⟨*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, **`BASEESAB`** is the longest palindrome substring of `BUB`**`BASEESAB`**`ANANA` ("Bubba sees a banana."). Thus, given the input string `BUBBASEESABANANA`, your algorithm should return the integer 8.

   (b) 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):

$$\texttt{BUB} + \texttt{BASEESAB} + \texttt{ANANA}$$
$$\texttt{B} + \texttt{U} + \texttt{BB} + \texttt{A} + \texttt{SEES} + \texttt{ABA} + \texttt{NAN} + \texttt{A}$$
$$\texttt{B} + \texttt{U} + \texttt{BB} + \texttt{A} + \texttt{SEES} + \texttt{A} + \texttt{B} + \texttt{ANANA}$$
$$\texttt{B} + \texttt{U} + \texttt{B} + \texttt{B} + \texttt{A} + \texttt{S} + \texttt{E} + \texttt{E} + \texttt{S} + \texttt{A} + \texttt{B} + \texttt{A} + \texttt{N} + \texttt{A} + \texttt{N} + \texttt{A}$$

Describe and analyze an algorithm to find the smallest number of palindromes that make up a given input string. For example:

- Given the string BUBBASEESABANANA, your algorithm should return the integer 3.
- Given the string PALINDROME, your algorithm should return the integer 10.
- Given the string RACECAR, your algorithm should return the integer 1.

(c) A ***metapalindrome*** is a decomposition of a string into a sequence of non-empty palindromes, such that the sequence of palindrome lengths is itself a palindrome. For example, the decomposition

$$\texttt{BUB} \bullet \texttt{B} \bullet \texttt{ALA} \bullet \texttt{SEES} \bullet \texttt{ABA} \bullet \texttt{N} \bullet \texttt{ANA}$$

is a metapalindrome for the string BUBBALASEESABANANA, with the palindromic length sequence $(3, 1, 3, 4, 3, 1, 3)$. Describe and analyze an efficient algorithm to find the length of the shortest metapalindrome for a given string. For example:

- Given the string BUBBALASEESABANANA, your algorithm should return the integer 7.
- Given the string PALINDROME, your algorithm should return the integer 10.
- Given the string DEPOPED, your algorithm should return the integer 1.

4. ***⟨⟨F16⟩⟩*** 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 $Wait[1..n]$.

5. ***⟨⟨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.

6. ⟨⟨*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.

$$^{\texttt{BANANA}}{}_{\texttt{ANANAS}} \qquad ^{\texttt{BAN}}{}_{\texttt{ANA}}{}^{\texttt{ANA}}{}_{\texttt{NAS}} \qquad ^{\texttt{B}}{}_{\texttt{AN}}{}^{\texttt{AN}}{}_{\texttt{A}}{}^{\texttt{A}}{}_{\texttt{NA}}{}^{\texttt{NA}}{}_{\texttt{S}}$$

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

$$\texttt{PRO}^{\texttt{D}}\texttt{G}^{\texttt{Y}}\texttt{R}^{\texttt{NAM}}\texttt{AMMI}^{\texttt{I}}\texttt{N}^{\texttt{C}}\texttt{G} \qquad {}^{\texttt{DY}}\texttt{PRO}^{\texttt{N}}\texttt{G}^{\texttt{A}}\texttt{R}^{\texttt{M}}\texttt{AMM}^{\texttt{IC}}\texttt{ING}$$

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.

7. 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:

$$32492 \to \overset{57^2}{3249} \to \overset{18^2}{324} \to 24 \to \overset{2^2}{4} \to \varepsilon.$$

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 $O(k^2)$ time***.

8. Suppose you are given a sequence of non-negative integers separated by $+$ and $\times$ 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 $\times$ 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 $\times$ if i is odd. Assume any arithmetic operation in your algorithm takes $O(1)$ time.

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. (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.

11. *⟨⟨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:

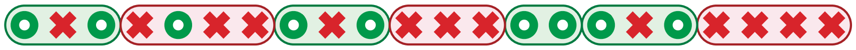
    $$\Big[\ \overbrace{1, 6, -1, 8, 0, 3, 3, 9, 8}^{37}\ \Big|\ \overbrace{8, 7, 4, 9, 8}^{36}\ \Big|\ \overbrace{9, 4, 8, 4, 8, 2}^{35}\ \Big]$$

    The numbers above each interval show the sum of the values in that interval.

12. *⟨⟨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 $1, 2, \dots, n$. 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 $k = 2$ (so each district contains either 2, 3, or 4 addresses). 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.

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 eight-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 mobile 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*⟩⟩ 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.

18. ⟨⟨***F19***⟩⟩ Satya is in charge of establishing a new testing center for the Standardized Awesomeness Test (SAT), and he 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. 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.

    (a) 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.

    (b) 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. Analyze your algorithm as a function of both n and k.

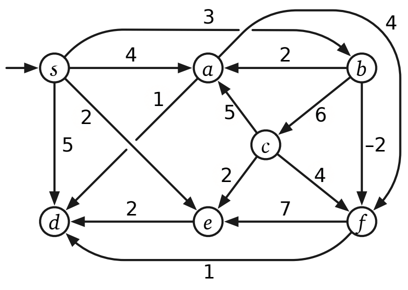
    Parts (a) and (b) appeared as complete problems in different versions of the same exam.

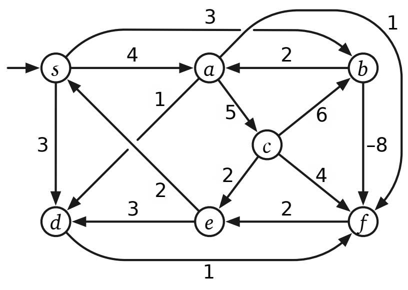
Graph Algorithms

Sanity Check

1. ⟨⟨*S14, F14, F16, F19*⟩⟩ Indicate the following structures in the example graphs below.

   - To indicate a subgraph (such as a path, a spanning tree, or a cycle), draw over every edge in the subgraph with a **heavy black line**. Your subgraph should be visible from across the room.
   - To indicate a subset of vertices, either draw a heavy black line around the entire subset, completely blacken the vertices in the subset, or list the vertex labels.
   - If the requested structure does not exist, just write the word NONE.





(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.

(e) The set of all vertices that can reach node c.

(f) The strong components. (Circle each strong component.)

(g) A simple cycle containing vertex s.

(h) A directed cycle with the minimum number of edges.

(i) A directed cycle with the smallest total weight.

(j) A walk from s to d with the maximum number of edges.

(k) A walk from s to d with the largest total weight.

(l) A depth-first pre-ordering of the vertices. (List the vertices in order.)

(m) A depth-first post-ordering of the vertices. (List the vertices in order.)

(n) A topological ordering of the vertices. (List the vertices in order.)

(o) A breadth-first ordering of the vertices. (List the vertices in order.)

(p) Draw the strong-component graph.

[On an actual exam, we would only ask about one graph, and we would ask for only a few of these structures. If the exam is given on paper, 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 vertex 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.)

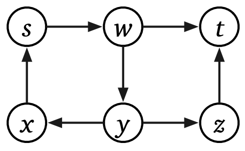
   (h) Find the *shortest* simple cycle, or correctly report that the graph has no cycles.

   (i) Determine whether deleting a given vertex v would disconnect the graph.

   [On an actual exam, we would ask for at most a few of 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.

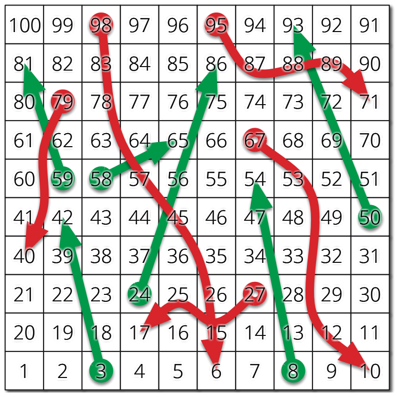
   For example, given the graph below, with the indicated vertices s and t, your algorithm should return TRUE, because the walk $s{\to}w{\to}y{\to}x{\to}s{\to}w{\to}t$ has length 6.



$$\begin{tikzcd}
s \arrow[r] & w \arrow[r] \arrow[d] & t \\
x \arrow[u] & y \arrow[l] \arrow[r] & z \arrow[u]
\end{tikzcd}$$

   [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 the graph $G = (V, E)$ and two vertices $u, v \in V$ (which may or may not be distinct).

   (b) *⟨⟨Lab⟩⟩* Now 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. The input to your algorithm consists of the graph $G = (V, E)$ and an array of 42 vertices (which may or may not be distinct). 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.

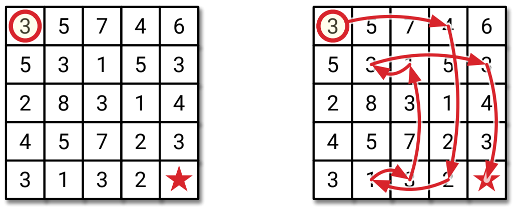
   (a) Prove that every tree is a bipartite graph.

   (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,

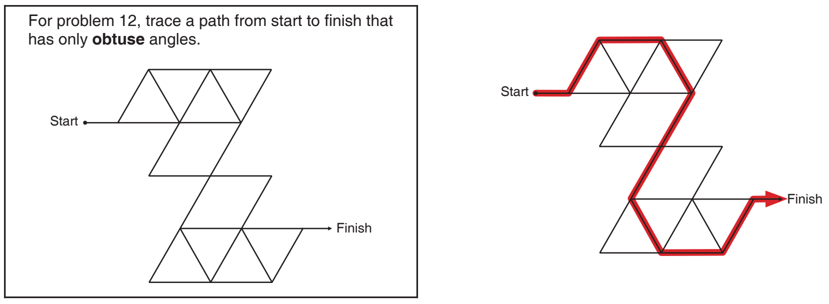
you are allowed to move the token up, down, left, or right; the distance you may move the token is determined by the number on its current square. For example, if the token is on a square labeled 3, then you may move the token three steps up, three steps down, three steps left, or three steps right. 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.



A 5×5 number maze that can be solved in eight moves.

7. ⟨⟨*F16*⟩⟩ The following puzzle appeared in my daughter's math workbook several years ago.[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.
Start
Finish
Start
Finish

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.

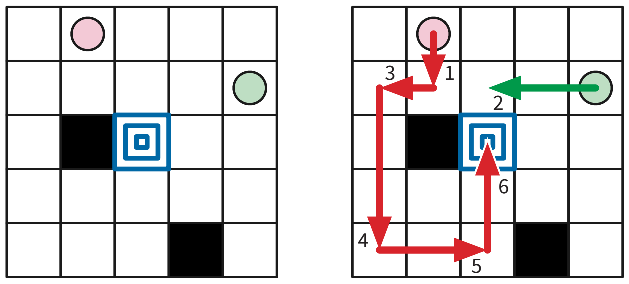
Your algorithm should return TRUE if G contains a walk from Start to Finish that has only obtuse angles, and FALSE otherwise. Formally, a walk through G is valid if $\pi/2 < \angle uvw \leq \pi$ for every pair of consecutive edges $u \rightarrow v \rightarrow w$ in the walk. Assume you have a subroutine that can determine whether the angle between any two segments is acute, right, obtuse, or straight in $O(1)$ time.

8. The famous puzzle-maker Kaniel the Dane invented a solitaire game played with two tokens on an $n \times n$ square grid. Some squares of the grid are marked as *obstacles*, and one

[1]Jason Batterson and Shannon Rogers, *Beast Academy Math: Practice 3A*, 2012. See https://www.beastacademy.com/resources/printables.php for more examples.

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. *[Hint: Don't forget about the time required to construct the graph!]*

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.

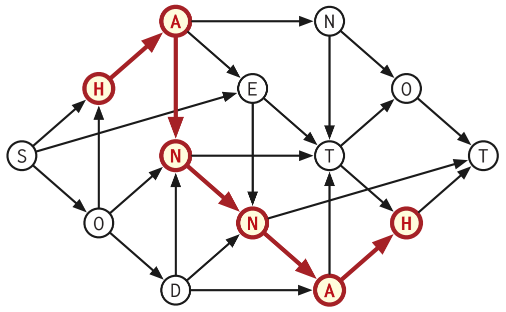
   (f) Describe an algorithm to compute the number of distinct paths from s to t in G. (Assume that you can add arbitrarily large integers in $O(1)$ time.)

3. Suppose you are given a directed acyclic graph G whose nodes represent jobs and whose edges represent *precedence constraints*: Each edge $u \to v$ indicates that job u must be completed before job v begins. Each node v stores a non-negative number $v.duration$ indicating the time required to execute job v. All jobs are executed in parallel; any job can start or end while any number of other jobs are executing, provided all the precedence constraints are satisfied. You'd like to get all these jobs done as quickly as possible.

   Describe an algorithm to determine, for every vertex v in G, the earliest time that job v can ***begin***, assuming the first job starts at time 0 and no precedence constraints are violated. Your algorithm should record the answer for each vertex v in a new field $v.earliest$.

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.

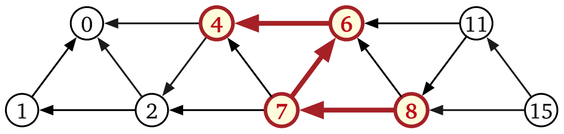
   Describe and analyze an algorithm to find the length of the longest palindrome that is the label of a path in G. For example, given the dag below, your algorithm should return the integer 6, which is the length of the palindrome HANNAH.



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 $h(u) - h(v)$.

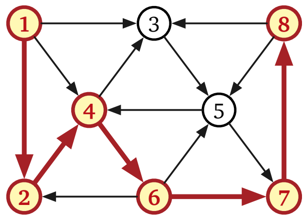
   Describe and analyze an algorithm to find the ***minimum span*** of a path in G with ***at least*** k edges. Your input consists of the graph G, the vertex heights $h(\cdot)$, and the integer k. Report the running time of your algorithm as a function of V, E, and k.

   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 $8 \to 7 \to 6 \to 4$.



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 $1 \to 2 \to 4 \to 6 \to 7 \to 8$.



Shortest Paths

1. Suppose you are given a directed graph G with weighted edges and a vertex s of G.

   (a) ⟨⟨*F14*⟩⟩ 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.

   (b) Suppose every vertex v stores a finite real value $dist(v)$. (In particular, $dist(v)$ is never equal to ∞ or $-\infty$.) Describe and analyze an algorithm to determine whether these real values are correct shortest-path distances from 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.

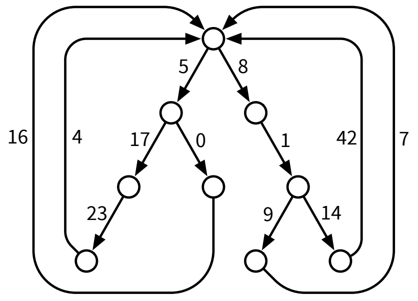


Figure 2. A looped tree.

(a) How much time would Dijkstra's algorithm require to compute the shortest path from one vertex s to another vertex t 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.)

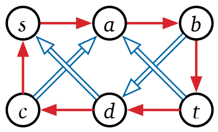
7. When there is more than one shortest path from one node s to another node t, it is often convenient to choose a shortest path with the fewest edges; call this the ***best*** path from s to t. Suppose we are given a directed graph G with positive edge weights and a source vertex s in G. Describe and analyze an algorithm to compute best paths in G from s to every other vertex.

8. 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.

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.

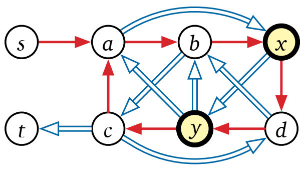
   For example, if you are given the graph below (where single arrows are red and double arrows are blue), your algorithm should return the integer 7, because the shortest legal walk from s to t is $s{\to}a{\to}b{\Rightarrow}d{\to}c{\Rightarrow}a{\to}b{\to}c$.



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]

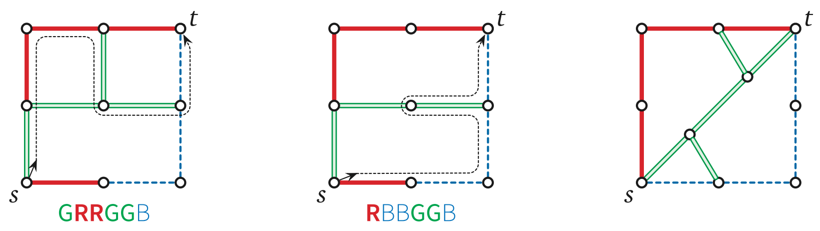
    For example, if you are given the following graph below as input (where single arrows are red, double arrows are blue), with special vertices x and y, your algorithm should return the integer 8, which is the length of the shortest legal walk $s{\to}x{\to}a{\to}b{\to}x{\Rightarrow}y{\Rightarrow}b{\Rightarrow}c{\Rightarrow}t$. The shorter walk $s{\to}a{\to}b{\Rightarrow}c{\Rightarrow}t$ is not legal, because vertex b is not special.



11. 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 ∞.

[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.



12. ⟨⟨*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.

13. ⟨⟨*F19*⟩⟩ 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.

14. ⟨⟨*F19*⟩⟩ 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, or correctly reports that no such path exists. Analyze your algorithm as a function of the parameters V, E, and k.

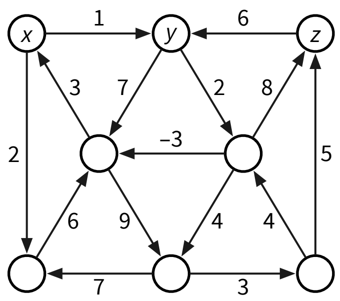
15. ⟨⟨*F19*⟩⟩ You and your friends are planning a hiking trip in Jellystone National Park over winter break. You have a map of the park's trails that lists all the scenic views in the park but also warns that certain trail segments have a high risk of bear encounters. To make the hike worthwhile, you 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 $t \in V$.

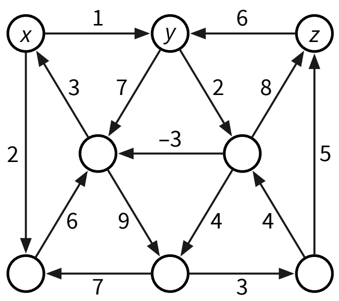
    Describe and analyze an algorithm to compute the shortest hike from your campsite s to the visitor center t that includes *at least* three scenic trail segments and *at most* one high-bear-risk trail segment. You may assume such a hike exists.

CS/ECE 374 A ✧ Fall 2021	Name:
Fake Midterm 2 Problem 1	

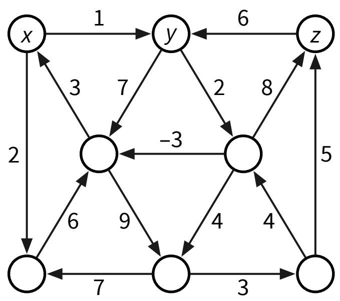
Clearly indicate the following structures in the directed graph below, or write NONE if the indicated structure does not exist. Don't be subtle; to indicate a collection of edges, draw a heavy black line along the entire length of each edge.



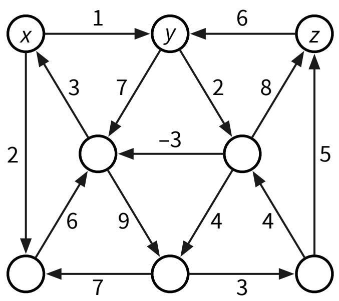
(a) A depth-first tree rooted at x.



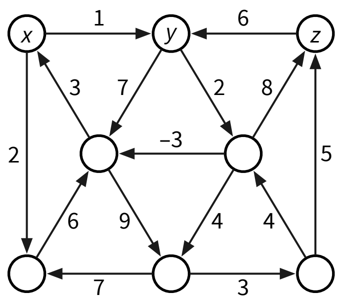
(b) A breadth-first tree rooted at y.



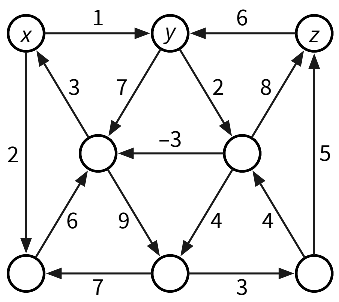
(c) A shortest-path tree rooted at z.



(d) The shortest directed cycle.



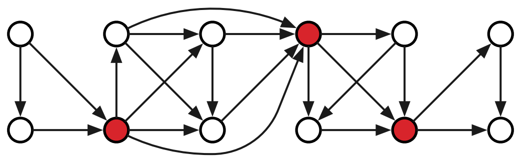
[scratch]



[scratch]

CS/ECE 374 A ✧ Fall 2021 Fake Midterm 2 Problem 2	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 2021 Fake Midterm 2 Problem 3	Name:

You decide to take your next hiking trip in Jellystone National Park. 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 along 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 $S \cap B = \emptyset$. 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 trail segments and *at most* one high-bear-risk trail segment. You may assume such a hike exists.

CS/ECE 374 A ✧ Fall 2021 **Fake Midterm 2 Problem 4**	Name:

During a family reunion over Thanksgiving break, your ultra-competitive thirteen-year-old nephew Elmo challenges you to a card game. At the beginning of the game, Elmo deals a long row of cards. Each card shows a number of points, which could be positive, negative, or zero. After the cards are dealt, you and Elmo alternate taking either the leftmost card or the rightmost card from the row, until all the cards are gone. The player that collects the most points is the winner.

For example, is the initial card values are $[4, 6, 1, 2]$, the game might proceed as follows:

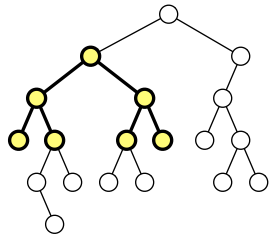
- You take the 4 on the left, leaving $[6, 1, 2]$.
- Elmo takes the 6 on the left, leaving $[1, 2]$.
- You take the 2 on the right, leaving $[1]$.
- Elmo takes the last 1, ending the game.
- You took $4 + 2 = 6$ points, and Elmo took $6 + 1 = 7$ points, so Elmo wins!

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. Assume that Elmo generously lets you move first.

CS/ECE 374 A ✧ Fall 2021	Name:
Fake Midterm 2 Problem 5	

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.



CS/ECE 374 A ✧ Fall 2021
Midterm 2
November 8, 2021

Directions

- ***Don't panic!***
- If you brought anything except your writing implements, your hand-written double-sided 8½" × 11" cheat sheet, please put it away for the duration of the exam. In particular, please turn off and put away *all* medically unnecessary electronic devices.
- **We *strongly* recommend reading the entire exam before trying to solve anything.** If you think a question is unclear or ambiguous, please ask for clarification as soon as possible.
- The exam has five numbered questions, each worth 10 points. (Subproblems are not necessarily worth the same number of points.)
- Write your answers on blank white paper using a dark pen. Please start your solution to each numbered question on a new sheet of paper.
- You have **120 minutes** to write your solutions, after which you have 30 minutes to scan your solutions, convert your scan to a PDF file, and upload your PDF file to Gradescope.
- If you are ready to scan your solutions before 9:00pm, send a private message to the host of your Zoom call ("Ready to scan") and wait for confirmation before leaving the Zoom call.
- Gradescope will only accept PDF submissions. Please do not scan your cheat sheets or scratch paper. Please make sure your solution to each numbered problem starts on a new page of your PDF file. **Low-quality scans will be penalized.**
- Proofs are required for full credit if and only if we explicitly ask for them, using the word ***prove*** in bold italics.
- Finally, if something goes seriously wrong, send email to jeffe@illinois.edu as soon as possible explaining the situation. If you have already finished the exam but cannot submit to Gradescope for some reason, include a complete scan of your exam **as a PDF file** in your email. If you are in the middle of the exam, send Jeff email, continue working until the time limit, and then send a second email with your completed exam **as a PDF file**. Please do not email raw photos.

---

1. Short answers:

   (a) Solve the recurrence $T(n) = 2T(n/3) + O(\sqrt{n})$.

   (b) Solve the recurrence $T(n) = 2T(n/7) + O(\sqrt{n})$.

   (c) Solve the recurrence $T(n) = 2T(n/4) + O(\sqrt{n})$.

   (d) Draw a connected undirected graph G with at most ten vertices, such that every vertex has degree at least 2, and no spanning tree of G is a path.

   (e) Draw a directed acyclic graph with at most ten vertices, exactly one source, exactly one sink, and more than one topological order.

   (f) Describe an appropriate memoization structure and evaluation order for the following (meaningless) recurrence, and give the running time of the resulting iterative algorithm to compute $Pibby(1, n)$. (Assume all array accesses are legal.)

$$Pibby(i,k) = \begin{cases} 0 & \text{if } i > k \\ A[i] & \text{if } i = k \\ Pibby(i+1,k-1) + A[i] + A[k] & \text{if } A[i] = A[k] \\ \max \left\{ \begin{array}{c} Pibby(i+2,k), \\ Pibby(i+1,k-1), \\ Pibby(i,k-2) \end{array} \right\} & \text{otherwise} \end{cases}$$

2. Your company has two offices, one in San Francisco and the other in New York. Each week you decide whether you want to work in the San Francisco office or in the New York office. Depending on the week, your company makes more money by having you work at one office or the other. You are given a schedule of the profits you can earn at each office for the next n weeks. You'd obviously prefer to work each week in the location with higher profit, but there's a catch: Flying from one city to the other costs \$1000. Your task is to design a travel schedule for the next n weeks that yields the maximum *total* profit, assuming you start in San Francisco.

   For example: suppose you are given the following schedule:

SF	\$800	\$200	\$500	\$400	\$1200
NY	\$300	\$900	\$700	\$2000	\$200

   If you spend the first week in San Francisco, the next three weeks in New York, and the last week in San Francisco, your total profit for those five weeks is $\$800 - \$1000 + \$900 + \$700 + \$2000 - \$1000 + \$1200 = \$3600$.

   (a) ***Prove*** that the obvious greedy strategy (each week, fly to the city with more profit) does not always yield the maximum total profit.

   (b) Describe and analyze an algorithm to compute the maximum total profit you can earn, assuming you start in San Francisco. The input to your algorithm is a pair of arrays $NY[1..n]$ and $SF[1..n]$, containing the profits in each city for each week.

3. Suppose you are given a directed graph $G = (V, E)$, whose vertices are either red, green, or blue. Edges in G do not have weights, and G is not necessarily a dag. The ***remoteness*** of a vertex v is the maximum of three shortest-path lengths:

   - The length of a shortest path to v from the closest red vertex
   - The length of a shortest path to v from the closest blue vertex
   - The length of a shortest path to v from the closest green vertex

   In particular, if v is not reachable from vertices of all three colors, then v is infinitely remote.

   Describe and analyze an algorithm to find a vertex of G whose remoteness is *smallest*.

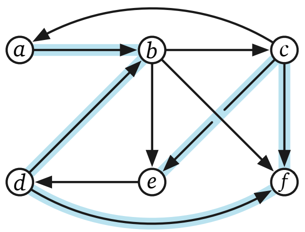
4. Suppose you are given an array $A[1..n]$ of integers such that $A[i] + A[i+1]$ is even for *exactly one* index i. In other words, the elements of A alternate between even and odd, except for exactly one adjacent pair that are either both even or both odd.

   Describe and analyze an efficient algorithm to find the unique index i such that $A[i] + A[i+1]$ is even. For example, given the following array as input, your algorithm should return the integer 6, because $A[6] + A[7] = 88 + 62$ is even. (Cells containing even integers are shaded blue.)

17	40	23	72	39	88	62	13	40	53	92	21	10	73	68

5. A *zigzag walk* in in a directed graph G is a sequence of vertices connected by edges in G, but the edges alternately point forward and backward along the sequence. Specifically, the first edge points forward, the second edge points backward, and so on. The *length* of a zigzag walk is the sum of the weights of its edges, both forward and backward.

   For example, the following graph contains the zigzag walk $a \rightarrow b \leftarrow d \rightarrow f \leftarrow c \rightarrow e$. Assuming every edge in the graph has weight 1, this zigzag walk has length 5.



   Suppose you are given a directed graph G with non-negatively weighted edges, along with two vertices s and t. Describe and analyze an algorithm to find the shortest zigzag walk from s to t in G.

CS/ECE 374 A ✧ Fall 2021

Conflict Midterm 2

November 9, 2021

Directions

- ***Don't panic!***
- If you brought anything except your writing implements, your hand-written double-sided 8½" × 11" cheat sheet, please put it away for the duration of the exam. In particular, please turn off and put away *all* medically unnecessary electronic devices.
- **We *strongly* recommend reading the entire exam before trying to solve anything.** If you think a question is unclear or ambiguous, please ask for clarification as soon as possible.
- The exam has five numbered questions, each worth 10 points. (Subproblems are not necessarily worth the same number of points.)
- Write your answers on blank white paper using a dark pen. Please start your solution to each numbered question on a new sheet of paper.
- You have **120 minutes** to write your solutions, after which you have 30 minutes to scan your solutions, convert your scan to a PDF file, and upload your PDF file to Gradescope.
- If you are ready to scan your solutions before 9:00pm, send a private message to the host of your Zoom call ("Ready to scan") and wait for confirmation before leaving the Zoom call.
- Gradescope will only accept PDF submissions. Please do not scan your cheat sheets or scratch paper. Please make sure your solution to each numbered problem starts on a new page of your PDF file. **Low-quality scans will be penalized.**
- Proofs are required for full credit if and only if we explicitly ask for them, using the word ***prove*** in bold italics.
- Finally, if something goes seriously wrong, send email to jeffe@illinois.edu as soon as possible explaining the situation. If you have already finished the exam but cannot submit to Gradescope for some reason, include a complete scan of your exam **as a PDF file** in your email. If you are in the middle of the exam, send Jeff email, continue working until the time limit, and then send a second email with your completed exam **as a PDF file**. Please do not email raw photos.

---

1. Short answers:

   (a) Solve the recurrence $T(n) = 3T(n/2) + O(n^2)$.

   (b) Solve the recurrence $T(n) = 7T(n/2) + O(n^2)$.

   (c) Solve the recurrence $T(n) = 4T(n/2) + O(n^2)$.

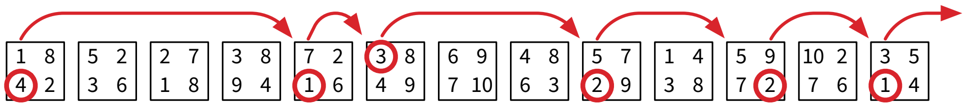
   (d) Draw a directed acyclic graph with at most ten vertices, exactly one source, exactly one sink, and more than one topological order.

   (e) Draw a directed graph with at most ten vertices, with distinct edge weights, that has more than one shortest path from some vertex s to some other vertex t.

   (f) Describe an appropriate memoization structure and evaluation order for the following (meaningless) recurrence, and give the running time of the resulting iterative algorithm to compute $Huh(1, n)$.

$$Huh(i,k) = \begin{cases} 0 & \text{if } i > n \text{ or } k < 0 \\ \min\begin{Bmatrix} Huh(i+1,k-2) \\ Huh(i+2,k-1) \end{Bmatrix} + A[i,k] & \text{if } A[i,k] \text{ is even} \\ \max\begin{Bmatrix} Huh(i+1,k-2) \\ Huh(i+2,k-1) \end{Bmatrix} - A[i,k] & \text{if } A[i,k] \text{ is odd} \end{cases}$$

2. *Quadhopper* is a solitaire game played on a row of n squares. Each square contains four positive integers. The player begins by placing a token on the leftmost square. On each move, the player chooses one of the numbers on the token's current square, and then moves the token that number of squares to the right. The game ends when the token moves past the rightmost square. The object of the game is to make as many moves as possible before the game ends.



A quadhopper puzzle that allows six moves. (This is ***not*** the longest legal sequence of moves.)

   (a) ***Prove*** that the obvious greedy strategy (always choose the smallest number) does not give the largest possible number of moves for every quadhopper puzzle.

   (b) Describe and analyze an efficient algorithm to find the largest possible number of legal moves for a given quadhopper puzzle.

3. Suppose you are given a directed graph $G = (V, E)$, each of whose vertices is either red, green, or blue. Edges in G do not have weights, and G is not necessarily a dag.

   Describe and analyze an algorithm to find a shortest path in G that contains at least one vertex of each color. (In particular, your algorithm must choose the best start and end vertices for the path.)

4. Your grandmother dies and leaves you her treasured collection of n radioactive Beanie Babies. Her will reveals that one of the Beanie Babies is a rare specimen worth 374 million dollars, but all the others are worthless. All of the Beanie Babies are equally radioactive, except for the valuable Beanie Baby, which is is either slightly more or slightly less radioactive, but you don't know which. Otherwise, as far as you can tell, the Beanie Babies are all identical.

   You have access to a state-of-the-art Radiation Comparator at your job. The Comparator has two chambers. You can place any two disjoint sets of Beanie Babies in Comparator's two chambers; the Comparator will then indicate which subset emits more radiation, or that the two subsets are equally radioactive. (Two subsets are equally radioactive if and only if they contain the same number of Beanie Babies, and they are all worthless.) The Comparator is slow and consumes a *lot* of power, and you really aren't supposed to use it for personal projects, so you *really* want to use it as few times as possible.

   Describe an efficient algorithm to identify the valuable Beanie Baby. How many times does your algorithm use the Comparator in the worst case, as a function of n?

5. Ronnie and Hyde are a professional robber duo who plan to rob a house in the Leverwood neighborhood of Sham-Poobanana. They have managed to obtain a map of the neighborhood in the form of a directed graph G, whose vertices represent houses, whose edges represent one-way streets.

   - One vertex s represents the house that Ronnie and Hyde plan to rob.
   - A set X of special vertices designate eXits from the neighborhood.
   - Each directed edge $u \rightarrow v$ has a non-negative weight $w(u \rightarrow v)$, indicating the time required to drive directly from house u to house v.
   - Thanks to Leverwood's extensive network of traffic cameras, speeding or driving backwards along any one-way street would mean certain capture.

   Describe and analyze an algorithm to compute the shortest time needed to exit the neighborhood, starting at house s. The input to your algorithm is the directed graph $G = (V, E)$, with clearly marked subset of exit vertices $X \subseteq V$, and non-negative weights $w(u \rightarrow v)$ for every edge $u \rightarrow v$.

CS/ECE 374 A ✧ Fall 2021
Final Exam Study Questions

This is a "core dump" of potential questions for the final exam. 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 or short-answer 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 statement below, write "YES" or "True" if the statement is ***always*** true and "NO" or "False" otherwise, and give a brief (at most one short sentence) explanation of your answer. **Assume $P \neq NP$.** If there is any other ambiguity or uncertainty about an answer, write "NO" or "False". For example:

- $x + y = 5$
  NO — Suppose $x = 3$ and $y = 4$.
- 3SAT can be solved in polynomial time.
  NO — 3SAT is NP-hard.
- If $P = NP$ then Jeff is the Queen of England.
  YES — The hypothesis is false, so the implication is true.

---

1. Which of the following are clear English specifications of a recursive function that could possibly be used to compute the edit distance between two strings $A[1..n]$ and $B[1..n]$?

   (a) $\mathit{Edit}(i, j)$ is the answer for i and j.

   (b) $\mathit{Edit}(i, j)$ is the edit distance between $A[i]$ and $B[j]$.

   (c) $$\mathit{Edit}[i,j] = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ \mathit{Edit}[i-1, j-1] & \text{if } A[i] = B[j] \\ \max\left\{\begin{array}{c} 1 + \mathit{Edit}[i, j-1] \\ 1 + \mathit{Edit}[i-1, j] \\ 1 + \mathit{Edit}[i-1, j-1] \end{array}\right\} & \text{otherwise} \end{cases}$$

   (d) $\mathit{Edit}[1..n, 1..n]$ stores the edit distances for all prefixes.

   (e) $\mathit{Edit}(i, j)$ is the edit distance between $A[i..n]$ and $B[j..n]$.

   (f) $\mathit{Edit}[i, j]$ is the value stored at row i and column j of the table.

   (g) $\mathit{Edit}(i, j)$ is the edit distance between the last i characters of A and the last j characters of B.

   (h) $\mathit{Edit}(i, j)$ is the edit distance when i and j are the current characters in A and B.

   (i) Iterate through both strings and update $\mathit{Edit}[\cdot, \cdot]$ at each character.

   (j) $\mathit{Edit}(i, j, k, l)$ is the edit distance between substrings $A[i..j]$ and $B[k..l]$.

   (k) *[I don't need an English description; my pseudocode is clear enough!]*

---

2. Which of the following statements is true for ***every*** directed graph $G = (V, E)$?

   (a) $E \neq \varnothing$.

   (b) Given the graph G as input, Floyd-Warshall runs in $O(E^3)$ time.

(c) If G has at least one source and at least one sink, then G is a dag.

(d) We can compute a spanning tree of G using whatever-first search.

(e) If the edges of G are weighted, we can compute the shortest path from any node s to any node t in $O(E \log V)$ time using Dijkstra's algorithm.

---

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

(a) L is non-empty.

(b) L is infinite.

(c) L contains the empty string ε.

(d) L^* is infinite.

(e) L^* is regular.

(f) L is accepted by some DFA if and only if L is accepted by some NFA.

(g) L is described by some regular expression if and only if L is rejected by some NFA.

(h) L is accepted by some DFA with 42 states if and only if L is accepted by some NFA with 42 states.

(i) If L is decidable, then L is infinite.

(j) If L is not decidable, then L is infinite.

(k) If L is not regular, then L is undecidable.

(l) If L has an infinite fooling set, then L is undecidable.

(m) If L has a finite fooling set, then L is decidable.

(n) If L is the union of two regular languages, then its complement $\overline{L}$ is regular.

(o) If L is the union of two regular languages, then its complement $\overline{L}$ is context-free.

(p) If L is the union of two decidable languages, then L is decidable.

(q) If L is the union of two undecidable languages, then L is undecidable.

(r) If $L \notin \mathrm{P}$, then L is not regular.

(s) L is decidable if and only if its complement $\overline{L}$ is undecidable.

(t) Both L and its complement $\overline{L}$ are decidable.

---

4. Which of the following statements are true for ***at least one*** language $L \subseteq \{0, 1\}^*$?

(a) L is non-empty.

(b) L is infinite.

(c) L contains the empty string ε.

(d) L^* is finite.

(e) L^* is not regular.

(f) L is not regular but L^* is regular.

(g) L is finite and L is undecidable.

(h) L is decidable but L^* is not decidable.

(i) L is not decidable but L^* is decidable.

(j) L is the union of two decidable languages, but L is not decidable.

(k) L is the union of two undecidable languages, but L is decidable.

(l) L is accepted by an NFA with 374 states, but L is not accepted by a DFA with 374 states.

(m) L is accepted by an DFA with 374 states, but L is not accepted by a NFA with 374 states.

(n) L is regular and $L \notin \mathrm{P}$.

(o) There is a Turing machine that accepts L.

(p) There is an algorithm to decide whether an arbitrary given Turing machine accepts L.

---

5. Which of the following languages over the alphabet $\{0, 1\}$ are ***regular***?

(a) $\{0^m 1^n \mid m \geq 0 \text{ and } n \geq 0\}$

(b) All strings with the same number of 0s and 1s

(c) Binary representations of all positive integers divisible by 17

(d) Binary representations of all prime numbers less than 10^{100}

(e) $\{ww \mid w \text{ is a palindrome}\}$

(f) $\{wxw \mid w \text{ is a palindrome and } x \in \{0, 1\}^*\}$

(g) $\{\langle M \rangle \mid M \text{ accepts a regular language}\}$

(h) $\{\langle M \rangle \mid M \text{ accepts a finite number of non-palindromes}\}$

---

6. Which of the following languages/decision problems are ***decidable***?

(a) $\varnothing$

(b) $\{0^n 1^{2n} 0^n 1^{2n} \mid n \geq 0\}$

(c) $\{ww \mid w \text{ is a palindrome}\}$

(d) $\{\langle M \rangle \mid M \text{ accepts } \langle M \rangle \bullet \langle M \rangle\}$

(e) $\{\langle M \rangle \mid M \text{ accepts a finite number of non-palindromes}\}$

(f) $\{\langle M \rangle \bullet w \mid M \text{ accepts } ww\}$

(g) $\{\langle M \rangle \bullet w \mid M \text{ accepts } ww \text{ after at most } |w|^2 \text{ steps}\}$

(h) Given an NFA N, is the language $L(N)$ infinite?

(i) CircuitSat

(j) Given an undirected graph G, does G contain a Hamiltonian cycle?

(k) Given encodings of two Turing machines M and M', is there a string w that is accepted by both M and M'?

---

7. Which of the following languages can be proved undecidable ***using Rice's Theorem***?

   (a) $\varnothing$
   (b) $\{0^n 1^{2n} 0^n 1^{2n} \mid n \geq 0\}$
   (c) $\{ww \mid w \text{ is a palindrome}\}$
   (d) $\{\langle M\rangle \mid M \text{ accepts an infinite number of strings}\}$
   (e) $\{\langle M\rangle \mid M \text{ accepts a finite number of strings}\}$
   (f) $\{\langle M\rangle \mid M \text{ accepts either } \langle M\rangle \text{ or } \langle M\rangle^R\}$
   (g) $\{\langle M\rangle \mid M \text{ accepts both } \langle M\rangle \text{ and } \langle M\rangle^R\}$
   (h) $\{\langle M\rangle \mid M \text{ does not accept exactly 374 palindromes}\}$
   (i) $\{\langle M\rangle \mid M \text{ accepts some string } w \text{ after at most } |w|^2 \text{ steps}\}$
   (j) $\{\langle M\rangle \bullet w \mid M \text{ rejects } w \text{ after at most } |w|^2 \text{ steps}\}$
   (k) Given the encodings of two Turing machines M and M', is there a string w that is accepted by both M and M'?

---

8. Recall the halting language $\text{HALT} = \{\langle M\rangle \bullet w \mid M \text{ halts on input } w\}$. Which of the following statements about its complement $\overline{\text{HALT}} = \Sigma^* \setminus \text{HALT}$ are true?

   (a) $\overline{\text{HALT}}$ is empty.
   (b) $\overline{\text{HALT}}$ is regular.
   (c) $\overline{\text{HALT}}$ is infinite.
   (d) $\overline{\text{HALT}}$ is in NP.
   (e) $\overline{\text{HALT}}$ is decidable.

---

9. Suppose some language $A \in \{0,1\}^*$ reduces to another language $B \in \{0,1\}^*$. Which of the following statements ***must*** be true?

   (a) A Turing machine that recognizes A can be used to construct a Turing machine that recognizes B.
   (b) A is decidable.
   (c) If B is decidable then A is decidable.
   (d) If A is decidable then B is decidable.
   (e) If B is NP-hard then A is NP-hard.
   (f) If A has no polynomial-time algorithm then neither does B.

---

10. Suppose there is a ***polynomial-time*** reduction from problem A to problem B. Which of the following statements ***must*** be true?

    (a) Problem B is NP-hard.
    (b) A polynomial-time algorithm for B can be used to solve A in polynomial time.
    (c) If B has no polynomial-time algorithm then neither does A.
    (d) If A is NP-hard and B has a polynomial-time algorithm then $\mathrm{P} = \mathrm{NP}$.
    (e) If B is NP-hard then A is NP-hard.
    (f) If B is undecidable then A is undecidable.

---

11. Consider the following pair of languages:

    - $\textsc{HamPath} := \big\{G \bigm| G \text{ is an undirected graph with a Hamiltonian path}\big\}$
    - $\textsc{Connected} := \big\{G \bigm| G \text{ is a connected undirected graph}\big\}$

    (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?

    (a) $\textsc{Connected} \in \mathrm{NP}$
    (b) $\textsc{HamPath} \in \mathrm{NP}$
    (c) $\textsc{HamPath}$ is decidable.
    (d) There is no polynomial-time reduction from $\textsc{HamPath}$ to $\textsc{Connected}$.
    (e) There is no polynomial-time reduction from $\textsc{Connected}$ to $\textsc{HamPath}$.

---

12. Consider the following pair of languages:

    - $\textsc{DirHamPath} := \big\{G \bigm| G \text{ is a directed graph with a Hamiltonian path}\big\}$
    - $\textsc{Acyclic} := \big\{G \bigm| G \text{ is a directed acyclic graph}\big\}$

    (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?

    (a) $\textsc{Acyclic} \in \mathrm{NP}$
    (b) $\textsc{Acyclic} \cap \textsc{DirHamPath} \in \mathrm{P}$
    (c) $\textsc{DirHamPath}$ is decidable.
    (d) There is a polynomial-time reduction from $\textsc{DirHamPath}$ to $\textsc{Acyclic}$.
    (e) There is a polynomial-time reduction from $\textsc{Acyclic}$ to $\textsc{DirHamPath}$.

---

13. Consider the following pair of languages:

    - $3\textsc{Color} := \big\{G \bigm| G \text{ is a 3-colorable undirected graph}\big\}$
    - $\textsc{Tree} := \big\{G \bigm| G \text{ is a connected acyclic undirected graph}\big\}$

(For concreteness, assume that in both of these languages, graphs are represented by adjacency matrices.) Which of the following ***must*** be true, assuming P≠NP?

(a) TREE is NP-hard.

(b) TREE ∩ 3COLOR ∈ P

(c) 3COLOR is undecidable.

(d) There is a polynomial-time reduction from 3COLOR to TREE.

(e) There is a polynomial-time reduction from TREE to 3COLOR.

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 tour*** in G is a closed walk W that visits every vertex of G exactly once, except for *at most one* vertex that W visits more than once.

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

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

4. An ***infra-Hamiltonian cycle*** in G is a closed walk W that visits every vertex of G exactly once, except for *at most one* vertex that W 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.

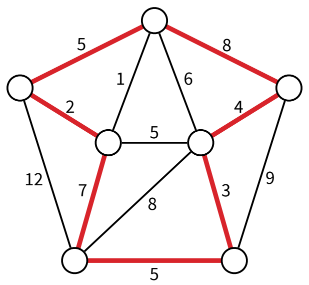
5. A ***quasi-satisfying assignment*** for a 3CNF boolean formula Φ is an assignment of truth values to the variables such that *at most one* clause in Φ does not contain a true literal. ***Prove*** that it is NP-hard to determine whether a given 3CNF boolean formula has a quasi-satisfying assignment.

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 Φ 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 subsets: 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 at least one guest who fits the description "was abducted by a flying saucer". 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 include at least one guest who "really enjoys 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 their involuntary flying-saucer voyage.

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 An Unofficial St. Brigid's Day party for the university staff.[1] His staff has a hierarchical structure; that is, the supervisor relation forms a directed, acyclic graph, with the president as the only source, and with 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 staff members 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)]

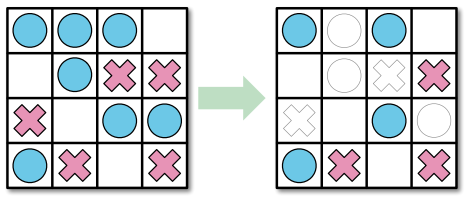
[1]As I'm sure you already know, St. Brigid of Kildare is one of the patron saints of Ireland, chicken and dairy farmers, and academics.

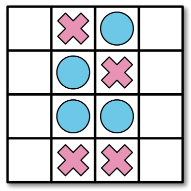
15. Consider the following solitaire game. The puzzle consists of an $n \times m$ grid of squares, where each square may be empty, occupied by a red stone, or occupied by a blue stone. The goal of the puzzle is to remove some of the given stones so that the remaining stones satisfy two conditions:

    (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.

16. To celebrate the end of the semester, Professor Jarling wants 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), Prof. Jarling 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 Prof. Jarling's problem is NP-hard.

17. 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?

18. 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?

19. 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.

20. 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 $\{0^n 1^n \mid n \geq 0\}$.

(c) A polynomial-time reduction from $\{0^n 1^n \mid n \geq 0\}$ 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. **The final exam will include a copy of this list.**

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?

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 or short-answer, but the following problems might still be useful to build intuition.

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. $\left\{ \mathtt{0}^n\mathtt{1}^n\mathtt{2}^n \;\middle|\; n \geq 0 \right\}$
3. $\left\{ A \in \{\mathtt{0},\mathtt{1}\}^{n\times n} \;\middle|\; n \geq 0 \text{ and } A \text{ is the adjacency matrix of a dag with } n \text{ vertices} \right\}$
4. $\left\{ A \in \{\mathtt{0},\mathtt{1}\}^{n\times n} \;\middle|\; n \geq 0 \text{ and } A \text{ is the adjacency matrix of a 3-colorable graph with } n \text{ vertices} \right\}$
5. $\left\{ \langle M \rangle \;\middle|\; M \text{ accepts } \langle M \rangle^R \right\}$
6. $\left\{ \langle M \rangle \;\middle|\; M \text{ accepts } \langle M \rangle^R \right\} \;\cap\; \left\{ \langle M \rangle \;\middle|\; M \text{ rejects } \langle M \rangle^R \right\}$
7. $\left\{ \langle M \rangle \mathtt{\#} w \;\middle|\; M \text{ accepts } ww^R \right\}$
8. $\left\{ \langle M \rangle \;\middle|\; M \text{ accepts } \mathtt{RICESTHEOREM} \right\}$
9. $\left\{ \langle M \rangle \;\middle|\; M \text{ rejects } \mathtt{RICESTHEOREM} \right\}$
10. $\left\{ \langle M \rangle \;\middle|\; M \text{ accepts at least one palindrome} \right\}$
11. $\Sigma^* \setminus \left\{ \langle M \rangle \;\middle|\; M \text{ accepts at least one palindrome} \right\}$
12. $\left\{ \langle M \rangle \;\middle|\; M \text{ rejects at least one palindrome} \right\}$
13. $\left\{ \langle M \rangle \;\middle|\; M \text{ accepts exactly one string of length } \ell, \text{ for each integer } \ell \geq 0 \right\}$
14. $\{ \langle M \rangle \mid \textsc{Accept}(M) \text{ has an infinite fooling set} \}$
15. $\left\{ \langle M \rangle \mathtt{\#} \langle M' \rangle \;\middle|\; \textsc{Accept}(M) \cap \textsc{Accept}(M') \neq \varnothing \right\}$
16. $\left\{ \langle M \rangle \mathtt{\#} \langle M' \rangle \;\middle|\; \textsc{Accept}(M) \oplus \textsc{Reject}(M') \neq \varnothing \right\}$ — 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.

$$\begin{aligned}
\textsc{SelfReject} &:= \big\{\langle M\rangle \bigm| M \text{ rejects } \langle M\rangle\big\}\\
\textsc{SelfAccept} &:= \big\{\langle M\rangle \bigm| M \text{ accepts } \langle M\rangle\big\}\\
\textsc{SelfHalt} &:= \big\{\langle M\rangle \bigm| M \text{ halts on } \langle M\rangle\big\}\\
\textsc{SelfDiverge} &:= \big\{\langle M\rangle \bigm| M \text{ does not halt on } \langle M\rangle\big\}
\end{aligned}$$

$$\begin{aligned}
\textsc{Reject} &:= \big\{\langle M\rangle\#w \bigm| M \text{ rejects } w\big\}\\
\textsc{Accept} &:= \big\{\langle M\rangle\#w \bigm| M \text{ accepts } w\big\}\\
\textsc{Halt} &:= \big\{\langle M\rangle\#w \bigm| M \text{ halts on } w\big\}\\
\textsc{Diverge} &:= \big\{\langle M\rangle\#w \bigm| M \text{ does not halt on } w\big\}
\end{aligned}$$

$$\begin{aligned}
\textsc{NeverReject} &:= \big\{\langle M\rangle \bigm| \textsc{Reject}(M) = \varnothing\big\}\\
\textsc{NeverAccept} &:= \big\{\langle M\rangle \bigm| \textsc{Accept}(M) = \varnothing\big\}\\
\textsc{NeverHalt} &:= \big\{\langle M\rangle \bigm| \textsc{Halt}(M) = \varnothing\big\}\\
\textsc{NeverDiverge} &:= \big\{\langle M\rangle \bigm| \textsc{Diverge}(M) = \varnothing\big\}
\end{aligned}$$

CS/ECE 374 A ✧ Fall 2021
Final Exam
December 15, 2021

Directions

- ***Don't panic!***
- If you brought anything except your writing implements, your two hand-written 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.
- **We *strongly* recommend reading the entire exam before trying to solve anything.** If you think a question is unclear or ambiguous, please ask for clarification as soon as possible.
- The exam has six numbered questions, each worth 10 points. (Subproblems are not necessarily worth the same number of points.)
- You have **150 minutes** to write your solutions, after which you have 30 minutes to scan your solutions, convert your scan to a PDF file, and upload your PDF file to Gradescope. (Both of these times are extended if you have time accommodations through DRES.)
- Proofs are required for full credit if and only if we explicitly ask for them, using the word ***prove*** in bold italics.
- Write your answers on blank white paper using a dark pen. Please start your solution to each numbered question on a new sheet of paper.
- If you are ready to scan your solutions and there are more than 15 minutes of writing time remaining, send a private message to the host of your Zoom call ("Ready to scan") and wait for confirmation before leaving the Zoom call.
- Gradescope will only accept PDF submissions. Please do not scan your cheat sheets or scratch paper. Please make sure your solution to each numbered problem starts on a new page of your PDF file.
- Finally, if something goes seriously wrong, send email to jeffe@illinois.edu as soon as possible explaining the situation. If you have already finished the exam but cannot submit to Gradescope for some reason, include a complete scan of your exam **as a PDF file** in your email. If you are in the middle of the exam, send Jeff email, continue working until the time limit, and then send a second email with your completed exam **as a PDF file**. Please do not email raw photos.

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?

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?

1. For each statement below, write "YES" if the statement is ***always*** true and "NO" otherwise, and give a ***brief*** (at most one short sentence) explanation of your answer. **Assume $P \neq NP$.** If there is any other ambiguity or uncertainty about an answer, write "NO". For example:

   - $x + y = 5$
     NO — Suppose $x = 3$ and $y = 4$.
   - 3SAT can be solved in polynomial time.
     NO — 3SAT is NP-hard.
   - If P = NP then Jeff is the Queen of England.
     YES — The hypothesis is false, so the implication is true.

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

Which of the following statements are true?

(a) The solution to the recurrence $T(n) = \mathbf{4}T(n/\mathbf{2}) + O(n^2)$ is $T(n) = O(n^2)$.

(b) The solution to the recurrence $T(n) = \mathbf{2}T(n/\mathbf{4}) + O(n^2)$ is $T(n) = O(n^2)$.

(c) Every directed acyclic graph contains at least one sink.

(d) Given *any* undirected graph G, we can compute a spanning tree of G in $O(V + E)$ time using whatever-first search.

(e) Suppose we want to iteratively evaluate the following recurrence:

$$What(i,j) = \begin{cases} 0 & \text{if } i > n \text{ or } j < 0 \\ \max\left\{ \begin{array}{c} What(i, j-1) \\ What(i+1, j) \\ A[i] \cdot A[j] + What(i+1, j-1) \end{array} \right\} & \text{otherwise} \end{cases}$$

We can fill the array $What[0..n, 0..n]$ in $O(n^2)$ time, by decreasing i in the outer loop and decreasing j in the inner loop.

Which of the following statements are true for ***at least one*** language $L \subseteq \{0, 1\}^*$?

(f) $L^* = (L^*)^*$

(g) L is decidable, but L^* is undecidable.

(h) L is neither regular nor NP-hard.

(i) L is in P, and L has an infinite fooling set.

(j) The language $\{\langle M \rangle \mid M \text{ accepts } L\}$ is undecidable.

---

2. For each statement below, write "YES" if the statement is ***always*** true and "NO" otherwise, and give a ***brief*** (at most one short sentence) explanation of your answer. **Assume *P ≠ NP*.** If there is any other ambiguity or uncertainty about an answer, write "NO".

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

   (Please remember to start your answers to this problem on a new page. Yes, this is really just a continuation of problem 1; we split it into two problems to make grading easier.)

Consider the following pair of languages:

- $\text{ACYCLIC} := \{\text{undirected graph } G \mid G \text{ contains no cycles}\}$
- $\text{HALFIND} := \{\text{undirected graph } G = (V, E) \mid G \text{ has an independent set of size } |V|/2\}$

(For concreteness, assume that in both of these languages, graphs are represented by their adjacency matrices.) The language HALFIND is actually NP-hard; **you do *not* need to prove that fact**.

Which of the following statements are true, assuming P ≠ NP?

(a) ACYCLIC is NP-hard.

(b) $\text{HALFIND} \setminus \text{ACYCLIC} \in \text{P}$
(Recall that $X \setminus Y$ is the subset of elements of X that are not in Y.)

(c) HALFIND is decidable.

(d) A polynomial-time reduction from HALFIND to ACYCLIC would imply P=NP.

(e) A polynomial-time reduction from ACYCLIC to HALFIND would imply P=NP.

Suppose there is a ***polynomial-time*** reduction from some language A over the alphabet $\{0, 1\}$ to some other language B over the alphabet $\{0, 1\}$. Which of the following statements are true, assuming P ≠ NP?

(f) A is a subset of B.

(g) If $B \in P$, then $A \in P$.

(h) If B is NP-hard, then A is NP-hard.

(i) If B is decidable, then A is decidable.

(j) If B is regular, then A is decidable.

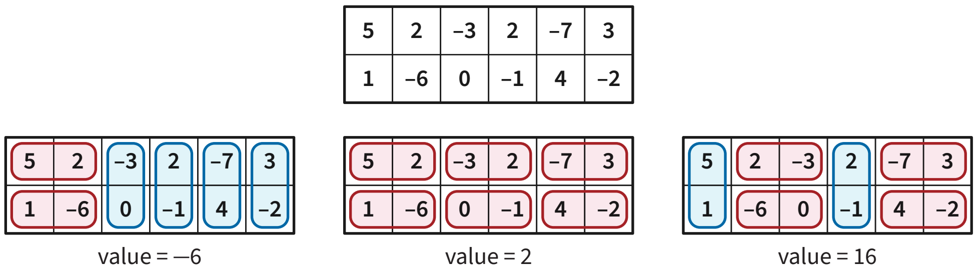
---

3. Suppose you are asked to tile a $2 \times n$ grid of squares with dominos (1×2 rectangles). Each domino must cover exactly two grid squares, either horizontally or vertically, and each grid square must be covered by exactly one domino.

   Each grid square is worth some number of points, which could be positive, negative, or zero. The ***value*** of a domino tiling is the sum of the points in squares covered by vertical dominos, *minus* the sum of the points in squares covered by horizontal dominos.

   Describe and analyze an efficient algorithm to compute the largest possible value of a domino tiling of a given $2 \times n$ grid. Your input is an array $Points[1..2, 1..n]$ of point values.

   As an example, here are three domino tilings of the same 2×6 grid, along with their values. The third tiling is optimal; no other tiling of this grid has larger value. Thus, given this 2×6 grid as input, your algorithm should return the integer 16.



4. Submit a solution to *exactly one* of the following problems. Don't forget to tell us which problem you've chosen!

   (a) Let Φ be a boolean formula in conjunctive normal form, with exactly three literals per clause (or in other words, an instance of 3SAT). ***Prove*** that it is NP-hard to decide whether Φ has a satisfying assignment in which *exactly half* of the variables are TRUE.

   (b) Let $G = (V, E)$ be an arbitrary undirected graph. Recall that a *proper 3-coloring* of G assigns each vertex of G one of three colors—red, blue, or green—so that every edge in G has endpoints with different colors. ***Prove*** that it is NP-hard to decide whether G has a proper 3-coloring in which *exactly half* of the vertices are red.

   (In fact, both of these problems are NP-hard, but we only want a proof for one of them.)

5. Suppose you are given a height map of a mountain, in the form of an $n \times n$ grid of evenly spaced points, each labeled with an elevation value. You can safely hike directly from any point to any neighbor immediately north, south, east, or west, but only if the elevations of those two points differ by at most Δ. (The value of Δ depends on your hiking experience and your physical condition.)

   Describe and analyze an algorithm to determine the longest hike from some point s to some other point t, where the hike consists of an uphill climb (where elevations must increase at each step) followed by a downhill climb (where elevations must decrease at each step). Your input consists of an array $Elevation[1..n, 1..n]$ of elevation values, the starting point s, the target point t, and the parameter Δ.

6. Recall that a ***run*** in a string $w \in \{0, 1\}^*$ is a maximal substring of w whose characters are all equal. For example, the string `00011111110000` is the concatenation of three runs:

$$\texttt{00011111110000} = \texttt{000} \bullet \texttt{1111111} \bullet \texttt{0000}$$

   (a) Let L_a denote the set of all strings in $\{0, 1\}^*$ where every `0` is followed immediately by at least one `1`.

   For example, L_a contains the strings `010111` and `1111` and the empty string ε, but does not contain either `001100` or `1111110`.

   - Describe a DFA or NFA that accepts L_a ***and***
   - Give a regular expression that describes L_a.

   (You do not need to prove that your answers are correct.)

   (b) Let L_b denote the set of all strings in $\{0, 1\}^*$ whose run lengths are increasing; that is, every run except the last is followed immediately by a *longer* run.

   For example, L_b contains the strings `0110001111` and `1100000` and `000` and the empty string ε, but does not contain either `000111` or `100011`.

   Prove that L_b is not a regular language.

CS/ECE 374 A ✧ Fall 2021

Conflict Final Exam

Directions

- ***Don't panic!***
- If you brought anything except your writing implements, your two hand-written 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.
- **We *strongly* recommend reading the entire exam before trying to solve anything.** If you think a question is unclear or ambiguous, please ask for clarification as soon as possible.
- The exam has six numbered questions, each worth 10 points. (Subproblems are not necessarily worth the same number of points.)
- You have **150 minutes** to write your solutions, after which you have 30 minutes to scan your solutions, convert your scan to a PDF file, and upload your PDF file to Gradescope. (Both of these times are extended if you have time accommodations through DRES.)
- Proofs are required for full credit if and only if we explicitly ask for them, using the word ***prove*** in bold italics.
- Write your answers on blank white paper using a dark pen. Please start your solution to each numbered question on a new sheet of paper.
- If you are ready to scan your solutions and there are more than 15 minutes of writing time, send a private message to the host of your Zoom call ("Ready to scan") and wait for confirmation before leaving the Zoom call.
- Gradescope will only accept PDF submissions. Please do not scan your cheat sheets or scratch paper. Please make sure your solution to each numbered problem starts on a new page of your PDF file.
- Finally, if something goes seriously wrong, send email to jeffe@illinois.edu as soon as possible explaining the situation. If you have already finished the exam but cannot submit to Gradescope for some reason, include a complete scan of your exam **as a PDF file** in your email. If you are in the middle of the exam, send Jeff email, continue working until the time limit, and then send a second email with your completed exam **as a PDF file**. Please do not email raw photos.

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?

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?

1. For each statement below, write "YES" if the statement is ***always*** true and "NO" otherwise, and give a ***brief*** (at most one short sentence) explanation of your answer. **Assume $P \neq NP$.** If there is any other ambiguity or uncertainty about an answer, write "NO". For example:

   - $x + y = 5$
     NO — Suppose $x = 3$ and $y = 4$.
   - 3SAT can be solved in polynomial time.
     NO — 3SAT is NP-hard.
   - If P = NP then Jeff is the Queen of England.
     YES — The hypothesis is false, so the implication is true.

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

Which of the following statements are true?

(a) The solution to the recurrence $T(n) = \mathbf{2}T(n/\mathbf{4}) + O(n^2)$ is $T(n) = O(n^2)$.

(b) The solution to the recurrence $T(n) = \mathbf{4}T(n/\mathbf{2}) + O(n^2)$ is $T(n) = O(n^2)$.

(c) For every directed graph G, if G has at least one source, then G has at least one sink.

(d) Given *any* undirected graph G, we can compute a spanning tree of G in $O(V + E)$ time using whatever-first search.

(e) Suppose we want to iteratively evaluate the following recurrence:

$$What(i,j) = \begin{cases} 0 & \text{if } i < 0 \text{ or } j < 0 \\ \max\left\{\begin{array}{c} What(i, j-1) \\ What(i-1, j) \\ A[i]\cdot A[j] + What(i-1, j-1) \end{array}\right\} & \text{otherwise} \end{cases}$$

We can fill the array $What[0..n, 0..n]$ in $O(n^2)$ time, by decreasing i in the outer loop and decreasing j in the inner loop.

Which of the following statements are true for ***all*** languages $L \subseteq \{0, 1\}^*$?

(f) $L^* = (L^*)^*$

(g) If L is decidable, then L^* is decidable.

(h) L is either regular or NP-hard.

(i) If L is undecidable, then L has an infinite fooling set.

(j) The language $\{\langle M \rangle \mid M \text{ decides } L\}$ is undecidable.

---

2. For each statement below, write "YES" if the statement is ***always*** true and "NO" otherwise, and give a ***brief*** (at most one short sentence) explanation of your answer. **Assume *P ≠ NP*.** If there is any other ambiguity or uncertainty about an answer, write "NO".

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

   (Please remember to start your answers to this problem on a new page. Yes, this is really just a continuation of problem 1; we split it into two problems to make grading easier.)

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 statements are true, assuming P ≠ NP?

(a) Acyclic ∈ NP

(b) Acyclic ∩ DirHamPath ∈ P

(c) DirHamPath is decidable.

(d) A polynomial-time reduction from DirHamPath to Acyclic would imply P=NP.

(e) A polynomial-time reduction from Acyclic to DirHamPath would imply P=NP.

Suppose there is a ***polynomial-time*** reduction from some language $A \subseteq \{0, 1\}$ reduces to some other language $B \subseteq \{0, 1\}$. Which of the following statements are true, assuming P ≠ NP?

(f) $A \subseteq B$.

(g) There is an algorithm to transform any Python program that solves B in polynomial time into a Python program that solves A in polynomial time.

(h) If A is NP-hard then B is NP-hard.

(i) If A is decidable then B is decidable.

(j) If a Turing machine M accepts B, the same Turing machine M also accepts A.

---

3. Aladdin and Badroulbadour are playing a cooperative game. Each player has an array of positive integers, arranged in a row of squares from left to right. Each player has a token, which starts at the leftmost square of their row; their goal is to move *both* tokens to the rightmost squares.

   On each turn, *both* players move their tokens *in the same direction*, either left or right. The distance each token travels is equal to the number under that token at the beginning of the turn. For example, if a token starts on a square labeled 5, then it moves either five squares to the right or five squares to the left. If *either* token moves past either end of its row, then both players immediately lose.

   For example, if Aladdin and Badroulbadour are given the arrays

A:	7	5	4	1	2	3	3	2	3	1	4	2
B:	5	1	2	4	7	3	5	2	4	6	3	1

   they can win the game by moving right, left, left, right, right, left, right. On the other hand, if they are given the arrays

A:	2	3	5	1	3
B:	3	4	1	2	1

   they cannot win the game. (The first move must be to the right; then Aladdin's token moves out of bounds on the second turn.)

   Describe and analyze an algorithm to determine whether Aladdin and Badroulbadour can solve their puzzle, given the input arrays $A[1..n]$ and $B[1..n]$.

4. Submit a solution to *exactly one* of the following problems. Don't forget to tell us which problem you've chosen!

   (a) Let $G = (V, E)$ be an arbitrary undirected graph. A subset $S \subseteq V$ of vertices is *mostly independent* if less than half the vertices of S have a neighbor that is also in S. ***Prove*** that finding the largest mostly independent set in G is NP-hard.

   (b) Let $G = (V, E)$ be an arbitrary directed graph with colored edges. A *rainbow Hamiltonian cycle* in G is a cycle that visits every vertex of G exactly one, in which no pair of consecutive edges have the same color. ***Prove*** that it is NP-hard to decide whether G has a rainbow Hamiltonian cycle.

   (In fact, both of these problems are NP-hard, but we only want a proof for one of them.)

5. 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:

$$32492 \to \overset{57^2}{3249} \to \overset{18^2}{324} \to 24 \to \overset{2^2}{4} \to \varepsilon.$$

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 $O(k^2)$ time***.

6. Recall that a ***run*** in a string $w \in \{0,1\}^*$ is a maximal substring of w whose characters are all equal. For example, the string `00011111110000` is the concatenation of three runs:

$$\texttt{00011111110000} = \texttt{000} \bullet \texttt{1111111} \bullet \texttt{0000}$$

(a) Let L_a denote the set of all strings in $\{0,1\}^*$ in which every run of `1`s has even length and every run of `0`s has odd length.

- Describe a DFA or NFA that accepts L_a ***and***
- Give a regular expression that describes L_a.

(You do not need to prove that your answers are correct.)

(b) Let L_b denote the set of all strings in $\{0,1\}^*$ in which every run of `0`s is immediately followed by a *longer* run of `1`s. ***Prove*** that L_b is *not* a regular language.

Both of these languages contain the strings `0111100011` and `110001111` and `111111` and the empty string ε, but neither language contains `000111` or `100011` or `0000`.