Final Exam
Two page cheat sheet
Conflict Mon 8-11
Structure
20pt T/F
6 x 10pt problems
1 x NP-hardness
2 x Midterm 1
2 x Midterm 2
1. Which of the following are a good English specifications of a recursive function that could possibly be used to compute the edit distance between two strings \(A[1..n]\) and \(B[1..n]\)?

   \(\text{Edit}(i, j)\) is the answer for \(i\) and \(j\).

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

   \[
   \text{Edit}[i, j] = \begin{cases} 
   i & \text{if } j = 0 \\
   j & \text{if } i = 0 \\
   \text{Edit}[i-1, j-1] & \text{if } A[i] = B[j] \\
   \min \left\{ \begin{array}{l} 
   1 + \text{Edit}[i-1, j-1] \\
   1 + \text{Edit}[i-1, j] \\
   1 + \text{Edit}[i, j-1] 
   \end{array} \right\} & \text{otherwise}
   \end{cases}
   \]

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

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

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

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

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

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

   \([I \ don't \ need \ an \ English \ description; \ my \ pseudocode \ is \ clear \ enough!]\)
(f) Suppose we want to prove that the following language is undecidable.

\[ \text{MUGGLE} := \{ \langle M \rangle \mid M \text{ accepts SCIENCE but rejects MAGIC} \} \]

Professor Potter, your instructor in Defense Against Models of Computation and Other Dark Arts, suggests a reduction from the standard halting language

\[ \text{HALT} := \{ \langle M, w \rangle \mid M \text{ halts on inputs } w \} . \]

Specifically, suppose there is a Turing machine \text{DETECTOMUGGLETUM} that decides MUGGLE. Professor Potter claims that the following algorithm decides HALT.

\[
\text{DECEDEHALT}(\langle M, w \rangle):
\]
Encode the following Turing machine:

\[
\text{RUBBERDuck}(x): \begin{align*}
\text{run } M & \text{ on input } w \\
\text{if } x &= \text{MAGIC} \\
\text{return } & \text{FALSE} \\
\text{else} \\
\text{return } & \text{TRUE}
\end{align*}
\]

return \text{DETECTOMUGGLETUM}(\langle \text{RUBBERDuck} \rangle)

Which of the following statements is true for all inputs \langle M, w \rangle?

- If \( M \) rejects \( w \), then \text{RUBBERDuck} rejects \text{MAGIC}.
- If \( M \) accepts \( w \), then \text{DETECTOMUGGLETUM} accepts \langle \text{RUBBERDuck} \rangle.
- If \( M \) rejects \( w \), then \text{DECEDEHALT} rejects \langle M, w \rangle.
- \text{DECEDEHALT} decides the language \text{HALT}. (That is, Professor Potter’s reduction is actually correct.)
- \text{DECEDEHALT} actually runs (or simulates) \text{RUBBERDuck}.
(d) Which of the following languages are *decidable*?

<table>
<thead>
<tr>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Binary representations of all perfect squares

\[ \{xy \in \{0,1\}^* \mid yx \text{ is a palindrome} \} \]

\[ \{\langle M \rangle \mid M \text{ accepts the binary representation of every perfect square} \} \]

\[ \{\langle M \rangle \mid M \text{ accepts a finite number of non-palindromes} \} \]

The set of all regular expressions that represent the language \(\{0,1\}^*\). (This is a language over the alphabet \(\{\emptyset, \varepsilon, 0, 1, *, +, (, )\}\).)

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

<table>
<thead>
<tr>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \{\langle M \rangle \mid M \text{ accepts a finite number of strings} \} \]

\[ \{\langle M \rangle \mid M \text{ accepts both } \langle M \rangle \text{ and } \langle M \rangle^R \} \]

\[ \{\langle M \rangle \mid M \text{ accepts exactly 374 palindromes} \} \]

\[ \{\langle M \rangle \mid M \text{ accepts some string } w \text{ after at most } |w|^2 \text{ steps} \} \]
2. A **quasi-satisfying assignment** for a 3CNF boolean formula \( \Phi \) is an assignment of truth values to the variables such that *at most one* clause in \( \Phi \) 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.

**Quasi 3SAT**

\[
(\overline{a} \lor b \lor c) \land (\overline{a} \lor c \lor \overline{v}) \land (\overline{b} \lor \overline{v} \lor \overline{d}) \land \ldots
\]

Given \( \Phi \) build \( \Phi' = \Phi \land (a \land b \land c) \land (\overline{a} \land b \land c) \land (\overline{a} \land c \land \overline{v}) \land (\overline{b} \land \overline{v} \land \overline{d}) \land \ldots \land (\overline{a} \land \overline{v} \land \overline{b} \land \overline{c}) \)

\( \Rightarrow \) If \( \Phi \) has sat assignment

- add \( a = b = c = \text{true} \)

  \( \rightarrow \) quasi sat assignment for \( \Phi' \)

  because all clauses in \( \Phi' \) ok

  exactly one new clause bad

\( \Leftarrow \) If \( \Phi' \) has quasi-sat assignment

- exactly one new clause bad

  so all old clauses good \( \rightarrow \) \( \Phi \) satisfied

Poly time
(a) Fix the alphabet $\Sigma = \{0, 1\}$. Describe and analyze an efficient algorithm for the following problem: Given an NFA $M$ over $\Sigma$, does $M$ accept at least one string? Equivalently, is $L(M) \neq \emptyset$?

(b) Recall from Homework 10 that deciding whether a given NFA accepts every string is NP-hard. Also recall that the complement of every regular language is regular; thus, for any NFA $M$, there is another NFA $M'$ such that $L(M') = \Sigma^* \setminus L(M)$. So why doesn’t your algorithm from part (a) imply that $P=NP$?
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 $A = [1, 6, -1, 8, 0, 3, 9, 8, 7, 4, 9, 8, 4, 8, 4, 8, 2]$ and the integer $k = 3$ as input, your algorithm should return the integer 37, which is the cost of the following partition:

$$\left[ \begin{array}{c|c|c} 37 & ? & 36 \\ 1, 6, -1, 8, 0, 3, 9, 8 & 8, 7, 4, 9, 8 & 9, 4, 8, 4, 8, 2 \end{array} \right]$$

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

Where is the next wall?

$$\text{MinCost}(i,k) = \begin{cases} \sum_{j=1}^{n} A[j] & \text{if } k = 1 \\ \min_{i \leq j \leq n+1} \left\{ \max_{k=1}^{n} \left\{ \sum_{j=1}^{k} A[j] \right\} + \text{MinCost}(j,k-1) \right\} & \text{if } i > n \\ \text{wall between } A[j-1] \text{ and } A[j] & \text{otherwise} \end{cases}$$

Memo: \[ O(nk) \text{ time} \]