Solutions to Homework 8

(10 pts) 1. **Self assessment.**

Carefully read and understand the posted solutions to the previous homework. Identify one part for which your own solution has the most room for improvement (e.g., has unsound reasoning, doesn't show what was required, could be significantly clearer or better organized, etc.). Copy or screenshot this solution, then in a few sentences, explain what was deficient and how it could be fixed.

(Alternatively, if you think one of your solutions is significantly *better* than the posted one, copy it here and explain why you think it is better.)

If you didn't do last week's homework, choose a problem from it that looks challenging to you, and in a few sentences, explain the key ideas behind its solution in your own words.

Solution: My answer for question 4 (a) was as follows:

> **Solution:** Suppose there is a polynomial-time mapping reduction from A to B via a function f. On input x:
>
> 1. Compute $f(x)$ in polynomial time.
> 2. Query the oracle for B on $f(x)$.
> 3. Accept if the oracle says "yes," and reject otherwise.
>
> This is a Turing reduction from A to B, so $A \leq_T B$.

This turing reduction is correct, but I forgot to include a correctness proof. This correctness proof would show that if $x \in A$, then $f(x) \in B$, and if $x \notin A$, then $f(x) \notin B$. Otherwise, my homework 7 had no other big issues.

2. **True, false, or unknown.**

   For each of the following statements, state, with brief justification, that it is *known to be true*, *known to be false*, or its truth/falsehood is *unknown*. If the statement is known to be false, either give a counterexample or briefly explain which part of the statement is false.

(4 pts) (a) If $\mathsf{P} \neq \mathsf{NP}$ and L is NP-complete, then $L \notin \mathsf{P}$.

> **Solution:** Known to be true; placing an NP-complete language in P implies $\mathsf{P} = \mathsf{NP}$.

(4 pts) (b) If $L \in \mathsf{NP}$, then $L \in \mathsf{P}$.

> **Solution:** Unknown; whether $\mathsf{P} = \mathsf{NP}$ or not remains an open question.

(4 pts) (c) Let V be an efficient verifier for some language $L \in \mathsf{NP}$. Let f be the Cook-Levin mapping reduction from L to SAT. The function $f(x)$ selects a certificate c, runs $V(x, c)$, and construct a Boolean formula ϕ from its computational tableau.

> **Solution:** Known to be false; the reduction cannot choose a specific certificate c because it only has access to the input x, not any valid witness.

3. **ϒ's alternative reductions.**

Professor ϒ is impressed by the polynomial-time mapping reductions presented in the past few lectures. However, ϒ thinks some of them have alternative solutions and presents to you here.

For each of the following reduction, identify the flaw(s) in the reduction and briefly explain why it fails. You may support your answers with counterexamples.

(5 pts) (a) In lecture, we proved that HAMCYCLE $\leq_p$ TSP, where

$$\text{HAMCYCLE} = \{G : G \text{ is an unweighted graph with a Hamiltonian cycle}\}$$
$$\text{TSP} = \{(G, k) : G \text{ is a weighted, complete graph with a tour of weight} \leq k\}\ .$$

ϒ notices that HAMCYCLE is itself decidable. They attempt to first find a Hamiltonian cycle in the graph and then set the weights of the edges in that cycle to 1, while assigning a weight of ∞ to all other edges. This ensures that the TSP instance has a tour of weight at most $|V|$ iff the original graph contains a Hamiltonian cycle.

Input: An unweighted graph G
Output: A weighted graph G' and nonnegative integer k

```
1:  function f(G = (V, E))
2:      V' ← V
3:      Initialize G' to be a complete graph on V' and initialize all weights to ∞ in E'
4:      Initialize C to be an empty sequence of vertices
5:      for each permutation Π = (v_1, v_2, ..., v_n) of V do
6:          if (v_i, v_{i+1}) ∈ E for all i = 1, ..., n − 1 then
7:              C ← Π
8:              break
9:      for each adjacent pair (v_i, v_{i+1}) in C do
10:         Set the weight of (v_i, v_{i+1}) to be 1 in E'
11:     return (G' = (V', E'), k = |V|)
```

Solution:

- The construction attempts to locate a Hamiltonian cycle in G by enumerating permutations, which solves HAMCYCLE directly and is not known to run in polynomial time. A valid polynomial-time reduction cannot depend on having a solution to HAMCYCLE in order to build the new instance.
- If no Hamiltonian cycle is found, all edges receive weight ∞, producing no valid TSP tour. This incorrectly discards the possibility of constructing a valid polynomial-time instance when the original graph has no Hamiltonian cycle, and thus fails as a reduction.

(5 pts) (b) In lecture, we also proved that 3SAT $\leq_p$ VERTEX-COVER, where

$$\text{3SAT} = \{\phi : \phi \text{ is a satisfiable 3CNF formula}\}$$
$$\text{VERTEX-COVER} = \{(G, k) : G \text{ is a graph with a vertex cover of size (at most) } k\}.$$

Υ thinks the reduction can be simplified. They propose that each variable "gadget" *should not have an edge between its two vertices.* Everything else about the reduction is left unchanged, i.e.,

- For each variable $x \in \phi$, create a variable gadget consisting of two vertices, respectively labeled by the literals x and $\overline{x}$, with *no edges between them.*
- For each clause $(\ell_i \vee \ell_j \vee \ell_k)$ in ϕ, create a clause gadget of three vertices, respectively labeled by these literals, with an edge between each pair of these vertices (i.e., a triangle).
- For each vertex of each clause gadget, create an edge between that vertex and the (unique) variable-gadget vertex having the same label.
- Finally, set $k = n + 2m$, where n, m are respectively the number of variables and clauses in ϕ.

Solution: Without an edge between each pair of variable-literal vertices, the construction no longer enforces having exactly one literal per variable. Both or neither literals can appear in the vertex cover, invalidating the intended one-to-one correspondence between vertex cover and truth assignment.

4. **Scheduling exams with flying colors.**

The EECS 376 staff email has been flooded with complaints about exam conflicts. To resolve this, they plan on working with the registrar's office to determine whether or not the exams can be scheduled so that no student has an exam conflict with another class.

As 376 staff, they have defined the problem as a language. Given a set of students S, a collection of subsets $C = \{C_1, \ldots, C_n\}$ of S where each C_i represents the students enrolled in course i, and the number of exam slots k, they want to know if each course can be assigned to an exam slot such that no student has two exams in the same slot.

Formally, the language SCHEDULING is defined as follows:

$$\text{SCHEDULING} = \{(S, C, k) : \exists f : C \to \{1 \ldots k\} \text{ s.t } f(C_i) = f(C_j) \implies C_i \cap C_j = \emptyset\}$$

As a star student in EECS 376, you recognize a potential challenge. The problem the staff has defined resembles a well-known computationally difficult problem. Save the day by proving that SCHEDULING is NP-hard!

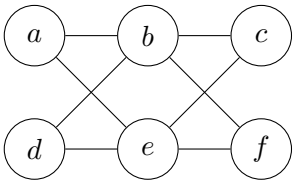
(6 pts) (a) You recall from the end of L17 that the Coloring Problem is NP-complete.[1] A graph is colorable by k colors if the vertices can be colored such that for any given edge (u, v), u and v have different colors. Formally, we can define it as follows:

$$\text{COLORING} = \{(G, k) : G \text{ is colorable by } k \text{ colors}\}.$$

Determine if each of the following is an instance of COLORING. If $(G, k) \in$ COLORING, give a coloring of the graph.[2] Otherwise, briefly explain why it's not possible.

[1] If interested, here's the NP-completeness proof for the 3-COLORING problem. With a little more work, it can be generalized to COLORING. You do not need to know the proof for this class, all you need is that 3-COLORING and COLORING are NP-complete.

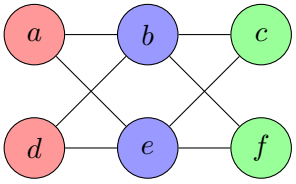
[2] You may give the vertex-color pairs, a table, etc. If you want to color the given graph in LaTeX, you can change the color of a node using `\node[fill=<color>]`.

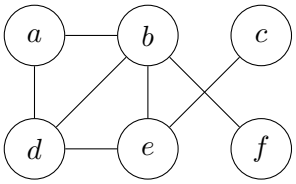
(i) $G =$ 

, $k = 3$

Solution:

$$a \mapsto 1, \quad b \mapsto 2, \quad c \mapsto 3, \quad d \mapsto 1, \quad e \mapsto 2, \quad f \mapsto 3.$$



(ii) $G =$ 

, $k = 2$

Solution: Not possible to 2-color, because there is an odd cycle (a, b, d).

(12 pts) (b) Show that COLORING $\leq_p$ SCHEDULING. Since COLORING is NP-hard, SCHEDULING is NP-hard.

Solution: Given (G, k) from COLORING with $G = (V, E)$, construct

$$S = E, \quad C = \{ C_v \mid v \in V \},$$

and assign each course $C_v \subseteq S$ by placing every edge in E incident to v into C_v. If k is the number of exam slots, define $f(G, k) = (S, C, k)$.

- ($\Rightarrow$) If G is colorable by k colors, color each vertex v by a color in $\{1, \ldots, k\}$, with a course slot for each color. Assign course C_v to the slot of the color of v. No adjacent vertices share a color, so no two courses sharing a student occupy the same slot.
- ($\Leftarrow$) If there is a valid assignment of courses to k slots, then no two courses sharing a student are placed in the same slot. Thus, no two adjacent vertices share a color in G. This yields a proper k-coloring.

Thus COLORING $\leq_p$ SCHEDULING, implying SCHEDULING is NP-hard.
Constructing S and C requires only a single pass through V and E, so f runs in polynomial time.

(12 pts) 5. **This recipe is bussin but NP-hard.**

We want to become a famous chef by creating a new dish. There are n ingredients, and we wish to use m of them ($m \leq n$) of them to create a new dish. There is an $n \times n$ matrix D that indicates the *discord* between two ingredients. In this case, each entry D_{ij} of the matrix D is an integer value between 0 and 10 (inclusive), where 0 means that the items i and j go together perfectly (there is no discord) and a 10 means they go together very badly. Any dish prepared with these ingredients is charged the sum of all pairs of ingredients as a penalty. If we can create a dish using at least m ingredients with a total penalty of at most p, the recipe is *bussin*.

Formally, this can be defined as:

$$\textsc{Bussin-Recipe} = \{(D, m, p) : D \in \{0 \dots 10\}^{n \times n}, \exists S \subseteq \{1 \dots n\} \text{ s.t } |S| \geq m \text{ and } \sum_{i,j \in S} D_{ij} \leq p\}$$

We assume for simplicity that $D_{ij} = D_{ji}$ and $D_{ii} = 0$ for all $i, j = 1, \dots n$.

Show that Bussin-Recipe is NP-hard.

Solution: Given an instance (G, k) of Clique with $G = (V, E)$ and $|V| = n$, construct $\big(D, m, p\big)$ as follows:

$$m \;=\; k, \quad p \;=\; 0, \quad D_{ij} \;=\; \begin{cases} 0 & \text{if } i = j \text{ or } (i,j) \in E, \\ 10 & \text{otherwise.} \end{cases}$$

- ($\Rightarrow$) If G has a clique of size k, then there is a set $S \subseteq \{1, \dots, n\}$ with $|S| \geq m$ and total penalty $\sum_{i,j \in S} D_{ij} = 0 \leq p$.
- ($\Leftarrow$) If there is a set S with $|S| \geq m$ and $\sum_{i,j \in S} D_{ij} \leq p$, then all pairs $(i, j) \in S$ must satisfy $D_{ij} = 0$, implying $(i, j) \in E$. Hence S is a clique of size k.

Thus Clique $\leq_p$ Bussin-Recipe, and Bussin-Recipe is NP-hard.

Building the matrix D requires checking each pair (i, j) in E; this is achievable in polynomial time.

6. **Subset sum.**

Recall that a multiset is just like a set, but can have duplicate elements. A submultiset also can have duplicate elements, as long as it does not have more copies of any specific element than its containing set does.[3]

Define the language

$$\text{SUBSET-SUM} = \{(A, s) : A \text{ is a multiset of integers} \geq 0, \text{ and } \exists\, I \subseteq A \text{ s.t } \sum_{a \in I} a = s\}.$$

In this problem you will prove that 3SAT $\leq_p$ SUBSET-SUM, thereby showing that SUBSET-SUM is NP-hard. Because SUBSET-SUM $\in$ NP (from HW 7), this implies that SUBSET-SUM is NP-complete.

[3] With a little more work, the version with ordinary sets can also be proved NP-complete.

The reduction is as follows. As input it is given a 3CNF formula ϕ with n variables $x_1, \ldots, x_n$ and m clauses $c_1, \ldots, c_m$. (Recall that each clause is the OR of exactly three literals, where a literal is either some variable x_i or its negation $\overline{x_i}$.)

The reduction constructs a collection of numbers as follows:

1. For each clause c_j, it constructs integers $a_j = b_j$, which are $m + n$ decimal digits long (counting any leading zeros), whose jth digits are 1, and whose other digits are 0.
2. For each variable x_i, it constructs integers t_i and f_i, each $m+n$ decimal digits long, where:
   (a) The $(m+i)$th digits of both t_i and f_i are 1.
   (b) The jth digit of t_i is 1 if literal x_i appears in clause c_j.
   (c) The jth digit of f_i is 1 if literal $\overline{x_i}$ appears in clause c_j.
   (d) All other digits of t_i and f_i are 0.
3. It constructs a target sum s that has $m + n$ decimal digits, where the first m digits are [**TODO: Part a**] and the last n digits are [**TODO: Part a**].

The reduction outputs (A, s), where A is the set of all the numbers a_j, b_j, t_i, and f_i.

For example, for the formula $\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$, the reduction constructs:

Number	$j=1$	$j=2$	$i=1$	$i=2$	$i=3$
a_1	1	0	0	0	0
b_1	1	0	0	0	0
a_2	0	1	0	0	0
b_2	0	1	0	0	0
t_1	1	0	1	0	0
f_1	0	1	1	0	0
t_2	0	1	0	1	0
f_2	1	0	0	1	0
t_3	1	0	0	0	1
f_3	0	1	0	0	1
s	?	?	?	?	?

The SUBSET-SUM instance for the given formula ϕ is therefore

$$(A = \{10000, 10000, 1000, 1000, 10100, 1100, 1010, 10010, 10001, 1001\}, s = \text{[TODO: Part b]}).$$

We highlight that the elements are written in *decimals*, so 10000 is the integer "ten thousand".

By inspection, this reduction can be seen to run in polynomial time.

(2 pts) (a) Give the complete step 3 from the description above. You do not need to justify your answer.

> **Solution:** It constructs a target sum s that has $m+n$ decimal digits, where the first m digits are **2** and the last n digits are **1**.

(4 pts) (b) Based on your answer in Part a, determine s in the example given above. Then, show that (A, s) is an instance of SUBSET-SUM by giving a subset of A that sums to s.

Solution: For $\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$: $m = 2$, $n = 3$, so

$$s = 22111.$$

$\{a_1, b_2, f_1, t_2, t_3\}$ sums to 22111, thus $(A, s) \in$ SUBSET-SUM.

(5 pts) (c) Using the reduction described above, construct the SUBSET-SUM instance corresponding to the 3SAT instance $\phi' = (x_1 \vee x_1 \vee x_1) \wedge (\overline{x_1} \vee \overline{x_1} \vee \overline{x_1})$. (Note that ϕ' is not satisfiable.) Also state whether the output instance is in the language SUBSET-SUM.

Solution: $\phi' = (x_1 \vee x_1 \vee x_1) \wedge (\overline{x_1} \vee \overline{x_1} \vee \overline{x_1})$:

$$A = \{a_1, b_1, t_1, f_1\}, \quad s = 21.$$

(A, s) is *not* in SUBSET-SUM, as there is no submultiset summing to 21.

(6 pts) (d) In the next two parts, you will prove that the reduction is correct. Show that

$$\phi \in \text{3SAT} \implies f(\phi) = (A, s) \in \text{SUBSET-SUM}.$$

That is, if the original 3SAT formula ϕ is satisfiable, then the corresponding SUBSET-SUM instance (A, s) is a "yes" instance.

Solution:
If $\phi \in$ 3SAT, then there exists a submultiset of A whose sum of decimal digits matches s. Thus, $(A, s) \in$ SUBSET-SUM.

(6 pts) (e) Prove the other direction of the correctness condition:

$$\phi \in \text{3SAT} \impliedby f(\phi) = (A, s) \in \text{SUBSET-SUM}.$$

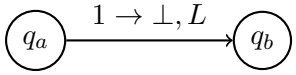
That is, if the constructed SUBSET-SUM instance (A, s) is a "yes" instance, then the original 3SAT formula ϕ is satisfiable.

Solution: If $\phi \in$ 3SAT, then there exists a submultiset of A whose sum of decimal digits matches s. Thus, $(A, s) \in$ SUBSET-SUM.

7. **Cook-Levin windows.**

Recall from lecture that one critical ingredient in the proof of the Cook-Levin theorem is to specify which contents of the 2×3 "windows" of the verifier's computation tableau are valid.

(6 pts) (a) Specify *all* the valid 2×3 window contents that could appear in two adjacent rows of the tableau, if those rows represent the verifier TM taking the following transition:



Since these windows are from two adjacent rows that represent taking the above transition, it cannot be the case, for example, that q_a appears in the bottom row of any of these windows.

For simplicity, you can assume that the symbols in the cells of the window are restricted to $\{0, 1, \perp, q_a, q_b\}$. Recall that there can be other symbols in the rows, such as the "boundary" symbol # and "separator" symbol \$, but we will ignore those here.

To avoid having to write out many similar windows, you may use one or more variables s_i to indicate that a cell could have any of 0, 1, or $\perp$. Such a variable, when repeated in multiple cells, represents the same symbol in every such cell. For example, instead of writing all three of the following windows (which may or may not be valid!):

0	0	1
0	0	$\perp$

0	1	1
1	0	$\perp$

0	$\perp$	1
$\perp$	0	$\perp$

you could simply write

0	s_1	1
s_1	0	$\perp$

.

If needed, you may use multiple variables $(s_1, s_2, \ldots)$ in a single window.

Be sure to include all valid window contents that represent the head of the TM being in *both* rows of the window, all those that represent the head being in *only one* of the rows of the window, and all those that represent the head not being in the window at all. Do not include any invalid windows.

You do not need to provide any justification.

Solution:

- *No head in either row:*

s_1	s_2	s_3
s_1	s_2	s_3

- *Head in the top row:*

q_a	s_1	s_2
$\perp$	s_1	s_2

s_1	q_a	s_2
q_b	$\perp$	s_2

s_1	s_2	q_a
s_1	q_b	$\perp$

- *Head in the bottom row:*

s_1	s_2	s_3
q_b	s_2	s_3

s_1	s_2	s_3
s_1	q_b	s_3

s_1	s_2	s_3
s_1	s_2	q_b

- *Head in both rows:*

s_1	s_2	q_a
s_1	q_b	$\perp$

(5 pts) (b) In the proof of Cook-Levin Theorem, we claimed that:

> If every 2-by-3 window is valid, and the first row is the start configuration, then each row of the tableau is a configuration that yields the next row.

Interestingly, the claim does not hold for 2-by-2 windows: it is possible for all 2×2 windows to be valid while the overall transition is invalid.

Give an example by first defining one or more Turing machine transitions, then providing two adjacent rows where each 2×2 window is valid, but the transition is incorrect.

Hint: consider an invalid pair of rows in which the active state of the head "disappears".

Solution: Define a Turing machine with a transition

$$\delta(q_c, 1) = (q_c, 1, R).$$

Consider these two adjacent rows of length 3:

$$\text{Top row:} \quad q_c \quad 1 \quad 1$$

$$\text{Bottom row:} \quad 1 \quad 1 \quad 1$$

Every 2×2 window in these rows is locally valid as no symbols conflict with the machine's rules. However, the machine's state q_c disappears incorrectly from the second row, making the overall transition invalid.

(4 pts) (c) Towards proving the Cook-Levin Theorem in lecture, we outlined a proof of the following statement:

> Let language $L \in \mathsf{NP}$ be arbitrary, and fix an efficient verifier VerifyL for L. Given any instance x of L, in polynomial time we can construct an instance ϕ of

SAT such that ϕ is satisfiable *if and only if* there *exists* a c such that the tableau of VerifyL(x, c) has q_{accept} in it.

Briefly explain why proving this statement proves that SAT is NP-hard.

Solution: Proving that the constructed ϕ is satisfiable if and only if there exists a c with q_{accept} in the tableau of VerifyL(x, c) establishes a PTMR from every language in NP to SAT, showing that SAT is NP-hard.