EECS 376: Foundations of Computer Science **Due 8:00pm, June 13**
University of Michigan, Spring 2024 **Homework 4** **(accepted until 9:59 pm, no credit after)**

This homework has 8 questions, for a total of 100 points and 0 extra-credit points.
Unless otherwise stated, each question requires *clear*, *logically correct*, and *sufficient* justification to convince the reader.
For bonus/extra-credit questions, we will provide very limited guidance in office hours and on Piazza, and we do not guarantee anything about the difficulty of these questions.
We strongly encourage you to typeset your solutions in LaTeX.
If you collaborated with someone, you must state their name(s). You must *write your own solution* for all problems and *may not use any other student's write-up*.

(0 pts) 0. **Before you start; before you submit.**

(a) Carefully read Handout 3 before starting this assignment, and apply it to the solutions you submit.

(b) If applicable, state the name(s) and uniqname(s) of your collaborator(s).

Solution: None.

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

Carefully read and understand the posted solutions to the **midterm exam**. Identify **one long-answer question** 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.

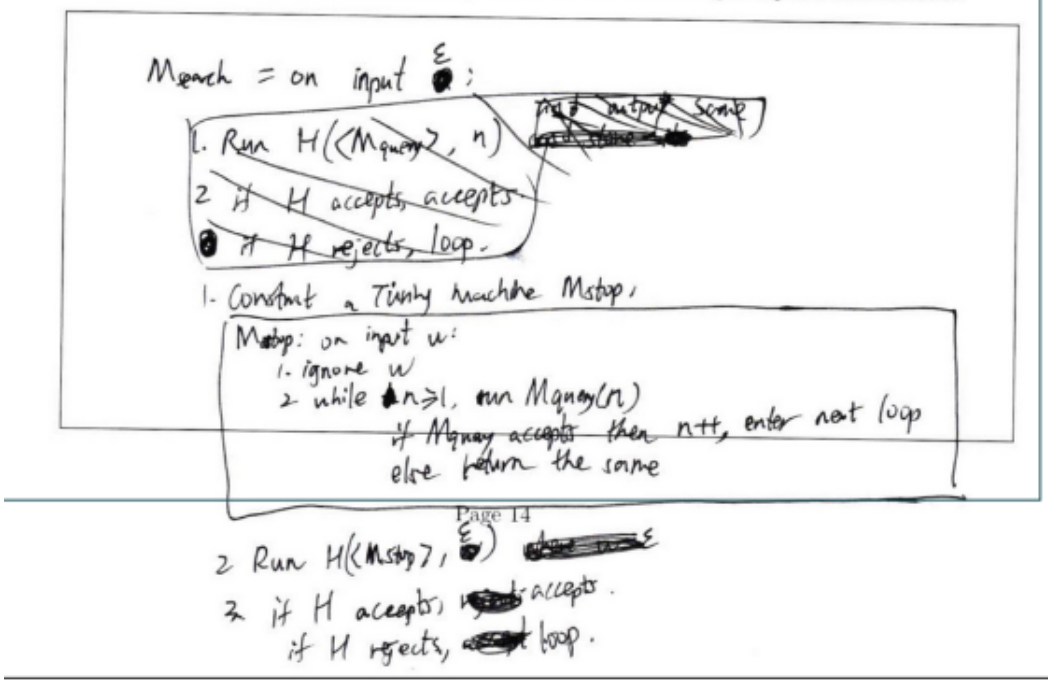
(Alternatively, if you think one of your solutions is significantly *better* than the posted one, copy it here and explain why you think it is better.)

You may reference the answer key, but your answer should be in your own words.

Solution:

(b) Suppose that L_{HALT} is decidable by a decider H.

Using H and M_{query} from (a), construct a Turing machine M_{search} that halts if there exists a natural number n for which iterating f starting from n does **not** reach 1 (a counterexample to the Collatz Conjecture), otherwise it loops indefinitely. **You do not need to justify its correctness.**



I did not work out the first long answer since I cannot think of the greediness inside the dp, so I think maybe I cannot improve that during exam. But I did work out the second long answer, while I did not get point for the (b) part, so I think I should have made it right if I were more careful. I makes this question more complex than it should be which is also wrong answer, since I did not consider carefully that if M_{query} did not halt on a natural number n, then it will loop infinitely, causing the H to reject, which cause the result to loop. Therefore in both way, the M_{stop} will loop and the H will reject, so the M_{search} will not halt on any input. Acctualy I do not need to construct any extra Turing Machine. It is fine to just iterate on natural number n and run $H(< M_{query} >, n)$ for each n.

2. P NP MAMC.

For each of the following multiple-answer multiple choice question, select all correct options. While justification is **not required** to received for full credit, providing it is highly encouraged. *Note: If you are editing the LaTeX file, change* `choice` *to* `CorrectChoice` *to fill in the bubble.*

(5 pts) (a) Let A and B be two languages. Suppose $A \leq_p B$, which of the following statement(s) is/are necessarily true?

- ◯ If A is in P then B is in P.
- ◯ If A is in P, then B is in NP.
- √ **If A is NP-Hard, then B is NP-Hard.**

√ **If A is NP-Complete, then B is NP-Hard.**
◯ Both A and B are in P or neither A nor B is in P.

Solution:

(5 pts) (b) Which of the following statements, if true, would imply P = NP?

◯ There exists a language in NP but that is not in P.
◯ Some NP language L is in P.
√ **SAT can be solved in $O(n^6)$ time.**
◯ There exists an NP-Complete language that is NP-Hard.
√ **$L_{\text{even}} = \{n : n$ is even$\}$ is NP-complete.**

Solution:

(5 pts) (c) Suppose P ≠ NP, which of the following statement(s) is/are true?

√ **NP-Complete languages cannot be decided efficiently.**
√ **If L is NP-Hard, then $L \notin$ P.**
◯ If $\overline{L}$ is NP-Hard, then L is not in P.
◯ Every language in the class NP is also NP-Hard.
√ **$L_{\text{odd}} = \{n : n$ is odd$\}$ is not NP-Complete.**

Solution:

3. **Understanding P, NP, and poly-time mapping reductions.**

(5 pts) (a) We claim that P ⊆ NP, i.e., for any language $L \in$ P, we have that $L \in$ NP as well. Here is an *incomplete* proof of this fact, which you will complete.

By the hypothesis that $L \in$ P, there is an efficient Turing machine M that decides L. We define the following efficient verifier V for L.

```
1: function V(x, c)
2:     [MISSING PSEUDOCODE]
```

State what the missing pseudocode should be, and prove that V is indeed an efficient verifier for L.

Solution:

```
1: function V(x, c)
2:     Ignore c
3:     Run M on input x
4:     if M accepts x then
5:         return TRUE
6:     else
7:         return FALSE
```

V is an efficient verifier for L because if $x \in L$, then M accepts x and so take some arbitrary dummit c we have $V(x, c)$ accepts x; if $x \notin L$ then M rejects x, so for all c we have $V(x, c)$ rejects x; And $V(x, c)$ runs in polynomial time since M runs in polynomial time.

(8 pts) (b) Show that poly-time mapping reductions are transitive. That is, if $A \leq_p B$ and $B \leq_p C$, then $A \leq_p C$.

Solution: Assume the hypothesis. Then there exists some function $f : \Sigma^* \to \Sigma^*$ such that for all $x \in \Sigma^*$, if $x \in A$ then $f(x) \in B$ and if $x \notin A$ then $f(x) \notin B$.
And there exists some function $g : \Sigma^* \to \Sigma^*$ such that for all $y \in \Sigma^*$, if $y \in B$ then $g(y) \in C$ and if $y \notin B$ then $g(y) \notin C$.
So now we consider the function $f \circ g$.
Let $x \in A$ be arbitrary, then $f(x) \in B$, so $g \circ f(x) = g(f(x)) \in C$.
Let $x \notin A$ be arbitrary, then $f(x) \notin B$, so $g \circ f(x) = g(f(x)) \notin C$.
This finishes the proof.

(8 pts) (c) Let $C = A \cup B$ where both A, B are languages in NP. State, with proof, whether C is in NP for *all*, *some* (but not all), or *no* such A, B.

Solution: Claim: $C = A \cup B$ is in NP, for all such A, B.
Proof:
Consider the following verifier for C:
Let V_A be a verifier for A and V_B be a verifier for B.
Input: $x \in A \cup B$, $c = (i, c')$ where $i \in 0, 1$ and c' is a certificate for either V_A or V_B.

```
1: function V(x, c)
2:     if i = 0 then run V_A(x, c')
3:     else run V_B(x, c')
```

if $x \in A \cup B$, then $x \in A$ or $x \in B$, so if $x \in A$ then there exists some c' such that $(0, c')$ verifies x in polynomial time, and if $x \in B$ then there exists some c' such that $(1, c')$ verifies x in polynomial time, since V_A and V_B run in polynomial time. If $x \notin C$, then $x \notin A$ and $x \notin B$, so for all c we have $V(x, c)$ rejects x.

4. **SAT Review.**

(5 pts) (a) Briefly explain what the "Boolean satisfiability" (SAT) problem is, and give one example instance that is satisfiable, and one that is unsatisfiable. Each example must include at least 2 variables and at least 3 (not necessarily distinct) logical operators (AND/OR/NOT).

Solution: "SAT" problem: Given a Boolean formula(which is made up of literals, ORs and ANDs, where literals means variables and their negations), output whether there is an assignment of truth values to the variables that makes the formula true.

Example of satisfiable instance: $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$, where $x_1 = T, x_2 = F$
Example of unsatisfiable instance: $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2)$ since whatever truth value we give to x_1 and x_2, there is always a false clause, so the formula is always false.

(5 pts) (b) 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: By proving this statement, we show that for any language L in NP, there exists some mapping f that maps every instace x of L to an instance ϕ in SAT, and $f(x) \in SAT$ iff $x \in L$, which by definition means that for arbitrary $L \in NP$, $L \leq_p SAT$, which by definition means that SAT is NP-Hard.

5. **The Greak Ham Trek.**

As a student at the University of Michigan, you're planning "The Great Ham Trek," a challenge to navigate from Burton Memorial Tower to the Michigan Union. The goal is to visit exactly k campus buildings, hosting mini-symposia on topics from ham radio innovation to the linguistics of ducks, without retracing your steps.

Formally, we define the following language:

$$\textsc{Ham-Trek} = \left\{ (G = (V, E), s, t, S, k) : \begin{array}{r} G \text{ has a simple path from } s \text{ to } t \text{ that} \\ \text{visits } \textbf{exactly } k \text{ of the vertices in } S \end{array} \right\}$$

Note: A *simple path* is a path without a cycle.

(8 pts) (a) Prove that Ham-Trek is efficiently verifiable, i.e., Ham-Trek $\in \mathsf{NP}$. Specifically, design and analyze (both correctness and runtime) an efficient verifier for Ham-Trek.

Solution: Consider the following verifier:
Input:$(G = (V, E), s, t, S, k, p)$ where G is a graph, $s, tinV$, $S \subseteq V$, k is an integer; and a path p in G as certificate.

```
1: function Verify(G, s, t, S, k, p)
2:     if k < 0 or k > |V| or then
3:         return FALSE
4:     if p[0] ≠ s or p[end] ≠ t then
5:         return FALSE
6:     visited ← set()
7:     count ← 0
```

```
8:    for i from 0 to length(p) − 1 do
9:        if (p[i], p[i − 1]) ∉ E then return FALSE
10:       if p[i] ∈ visited then return FALSE
11:       else
12:           visited.add(p[i])
13:       if p[i] ∈ S then
14:           count ← count + 1
15:   if count ≠ k then
16:       return FALSE
17:   return TRUE
```

Now we analyze the correctness and runtime of the verifier.
Correctness: If p is a path from s to t that visits exactly k vertices in S, then the verifier will return TRUE. If p is not a HamTrek, then either p is not a path from s to t, or it visits more or less than k vertices in S, or it contains a path that does not exists, or it turns back. In all such cases, the verifier will return FALSE.
Runtime: The verifier runs in $O(|V|^2)$ time, where $|V|$ is the length of the vertices set, since we iterate in the length of the path which is at most $|V|$, and in each iteration we iterate the visited vertice set which is at most as big as $|V|$ to confirm whether a vertice has been visited. Therefore the verifier is efficient.

(8 pts) (b) Prove that HAM-TREK is NP-Hard, by showing HAM-PATH $\leq_p$ HAM-TREK, where

$$\text{HAM-PATH} = \{(G, s, t) : G \text{ is a graph with a Hamiltonian path from } s \text{ to } t\}.$$

Note: A Hamiltonian path is a path that visits each vertex of the graph exactly once.

Solution: To show HAM-TREK is NP-Hard, we construct a polynomial-time reduction from HAM-PATH to HAM-TREK.
My construction: Given an instance $x = (G, s, t)$, map it to an instance for HAM-TREK as follows: Use the same graph G, vertice s and t. Set $S = V$, the set of all vertices in G. Set $k = |V|$. Then the constructed instance is $\phi(x) = (G, s, t, S, k)$. Now we prove the correctness of the polynomial-time reduction.
Correctness: If $x = (G, s, t) \in$ HAM-PATH, then by definition, there is a simple path visiting each vertex exactly once, thus visiting exactly $|V|$ vertices between s and t, which matches our k, so the constructed $\phi(x) = (G, s, t, S, k) \in$ HAM-TREK.

If the constructed instance $\phi(x) = (G, s, t, S, k) \in$ HAM-TREK, then there exists a simple path from s to t visiting exactly $|V| - 1$ vertices. Given that $S = V$, this path must visit each vertex exactly once, except possibly s or t once more, forming a Hamiltonian path, so $x \in Hampath$.

Constructing the HAM-TREK instance from a HAM-PATH instance involves only copying the graph, which is $O(|G|)$, in polynomial time with the input graph x.

Therefore $Hampath \leq_p$ HAM-TREK, and since $HamPath$ is NP-Hard, we have proved that HAM-TREK is NP-Hard.

(8 pts) 6. **More Knapsack!**

Recall the 0-1 knapsack problem: an instance is a list of n item weights $W = (W_1, W_2, \ldots, W_n)$, their corresponding values $V = (V_1, V_2, \ldots, V_n)$, and a weight capacity C. (All values are non-negative integers.) The goal is to select items having maximum total value, such that their total weight does not exceed the capacity.

We can make this a decision problem by introducing a "budget" K and asking whether a total value of at least K can be achieved (again, subject to the capacity constraint):

$$\text{KNAPSACK} = \{(W, V, C, K) : \exists\ S \subseteq \{1, \ldots, n\} \text{ such that } \sum_{i \in S} W_i \leq C \text{ and } \sum_{i \in S} V_i \geq K\}.$$

Prove that KNAPSACK is NP-Hard, by showing that SUBSETSUM $\leq_p$ KNAPSACK, where

$$\text{SUBSETSUM} = \{(A, s) : A \text{ is a } \textit{multiset} \text{ of integers} \geq 0, \text{ and } \exists\ I \subseteq A \text{ such that } \sum_{a \in I} a = s\}.$$

A *multiset* A is just like a set, but it can have duplicate elements. A submultiset $I \subseteq A$ also can have duplicate elements, as long as it does not have more copies of an particular element than A does.[1]

Solution:
Construction:
Input: (A, s) where A is a multiset of integers ≥ 0 and s is an positive integer.
Output:We take $W = V = A, C = K = s$, and take $\phi(A, s) = (W, V, C, K)$ as output.
Correctness:
If $(A, s) \in$ SUBSETSUM, then $s = C \leq C$ and $s = K \geq K$, so $\phi(x) \in$ KNAPSACK.
If $\phi(x) \in$ KNAPSACK, then some of the knapsack items has a total weight of $\leq C = s$ and a total value of $\geq K = s$. Since $W = V$, each item's weight equals its value, so the total weight and total value of these items are both s, which implies that there is a submultiset of A that sum up to s, so $(A, s) \in$ SUBSETSUM. And the construction of ϕ is efficient, since it only involves copying the input, which is $O(|A|)$, so SUBSETSUM $\leq_p$ KNAPSACK as needed. Since SUBSETSUM is NP-Hard, we have shown that KNAPSACK is NP-Hard.

7. **More Clique!**

[1] In class we defined a slightly different version of SUBSETSUM with ordinary sets instead of multisets. With a little more work, the version with ordinary sets can also be proved NP-Complete.

Recall that a *clique* of an undirected graph $G = (V, E)$ is a subset $C \subseteq V$ such that every pair of vertices in C has an edge between them. A clique with k vertices is refer to as a k-clique. In lecture, we proved that CLIQUE $\in$ NP, where

$$\text{CLIQUE} = \{(G = (V, E), k) : G \text{ has a clique of size} \geq k\}.$$

(8 pts) (a) A *triangle* in an undirected graph is a 3-clique. Consider the following variant of CLIQUE

$$\text{TRIANGLE} = \{G = (V, E) : G \text{ has a triangle.}\}$$

Show that TRIANGLE $\in$ P.

Solution: We construct a decider for TRIANGLE as follows:
Input: $G = (V, E)$ where G is a graph.

```
1: function TRIANGLE(G)
2:     for each v ∈ V do
3:         for each u ∈ V do
4:             if (v, u) ∈ E then
5:                 for each w ∈ V do
6:                     if (u, w) ∈ E and (v, w) ∈ E then
7:                         return TRUE
8:     return FALSE
```

Correctness: If G has a triangle, then there exists three vertices v, u, w such that $(v, u), (u, w), (v, w) \in E$, so the decider will return TRUE. If G does not have a triangle, then for all v, u, w we have $(v, u), (u, w), (v, w) \notin E$, so the decider will return FALSE.
Runtime:The decider runs in $O(|V|^3)$ time, since we iterate through all the vertices and for each pair of vertices we check if there is an edge between them, and if there is, we check if there is an edge between the third vertex and the other two vertices. Therefore the decider is efficient.
By the correctness and runtime analysis, we have shown that TRIANGLE $\in$ P.

(12 pts) (b) Now, consider another variant of CLIQUE:

$$\text{HALF-CLIQUE} = \{G = (V, E) : G \text{ has a clique of size} \geq |V|/2.\}$$

Prove that HALF-CLIQUE is NP-complete.

Solution:
Claim 1: Half-Clique $\in$ NP
Consider the following Verifier for HALF-CLIQUE:
Input: $(G = (V, E), p)$ where G is a graph and $p \subseteq V$ is a set of vertices in G.

```
1: function Verify(G, p)
2:     if |p| < |V|/2 then
```

```
        return FALSE
    for i = 0 to |p| − 1 do
        for j = i + 1 to |p| − 1 do
            if (p[i], p[j]) ∉ E then
                return FALSE
    return TRUE
```

Correctness: The verifier is correct since if $G \in$ Half-Clique, then there exists a clique of size at least $|V|/2$ in G, so by taking vertices in the clique as p, the verifier will return TRUE. If $G \notin$ Half-Clique, then there is no clique of size at least $|V|/2$ in G, so for all $p \subset V$, $Verify(G, p)$ return FALSE.
Efficiency: The verifier runs in polynomial time $O(|V|)^2$ since it iterates through all pairs of vertices in p for two nested loops, with the size of each at most $|V|$.

Above we have proved the correctness and efficiency of the verifier, so Half-Clique $\in$ NP.

Claim 2: Half-Clique $\in$ NP-Hard

We show that Half-Clique is $\mathsf{NP}-Hard$ by showing that Clique $\leq_p$ Half-Clique.
,
Construction:
Input: An instance of Clique $x = (G, k)$ specified by a graph $G = (V, E)$ and an integer k.
Output: A graph $\phi(x) = (V', E')$ as follows:
If $k \geq \frac{|V|}{2}$, then initialize $V' = V$ and add $2k - |V|$ vertices to V', so $|V'| = 2k$. Set $E' = E$, and return $\phi(x) = (V', E')$. This ensures that any clique of size k in G will be a clique of size $2k/2 = \frac{|V'|}{2}$ in G'.
If $k < \frac{|V|}{2}$, then initialize $V' = V$ and add $|V| - 2k$ new vertices to V, making $|V'| = 2|V| - 2k$. Then we connect these new vertices to each other and to all vertices in V, forming E'. This ensures that any clique of size k in G will be a clique of size $|V| - k = \frac{|V'|}{2}$ in G'.
Correctness:
1. If $x \in$ Clique, then there's a clique of size k in G. As stated in our construction, this clique together with the new vertices forms a clique of size at least $\frac{|V'|}{2}$, so $\phi(x) \in$ Half-Clique.
2. If $x \notin$ Clique, then G has at most s clique of size $k-1$. So by our construction, the new vertices will form a clique of size at most $\frac{|V'|-1}{2}$ in G', so $\phi(x) \notin$ Half-Clique.
Constructing V' and E' from V and E involves adding a linear number of vertices and edges relative to $|V|$, which is a $O(|V|)$ operation.
Thus, this polytime reduction is correct, showing that Clique $\leq_p$ Half-Clique, proving that Half-Clique is NP-Hard since Clique $\in$ NP-Hard.
Conclusion: Since Half-Clique $\in$ NP and Half-Clique is NP-Hard, we have shown that Half-Clique is NP-Complete.