Attach a fully filled-in cover sheet to the front of your printed homework. Your name should not appear anywhere; the cover sheet and each individual page of the homework should include your Banner ID only.

While collaboration is encouraged in this class, please remember not to take away notes from any labs or collaboration sessions.

Please monitor Piazza, as we will post clarifications of questions there. Hand in your solutions by 10:20am to the CSCI 1010 bin on the second floor of the CIT. Late homeworks are not accepted.

Problem 1

Note: You may want to wait until Lecture 16 (Tuesday, October 30th) before attempting this problem.

For each of the following $2 \times 3$ windows, decide whether it is:

(a) A legal window in a tableau regardless of the Turing machine’s transition function \( \delta \);

(b) A legal window in a tableau for certain \( \delta \) functions; or

(c) An illegal window in a tableau regardless of the Turing machine’s \( \delta \) function.

Assume that \( \Gamma = \{ x, y, z, \_ \} \). As an example, consider the following window:

\[
\begin{array}{ccc}
q_0 & x & y \\
\_ & y & q_1 \\
z & y & \_ \\
\end{array}
\]

This is always illegal because the tape head moves more than one space.
Problem 2

Prove that:

1. P is closed under union: if $A \in P$ and $B \in P$ then $A \cup B \in P$.

2. P is closed under concatenation: if $A \in P$ and $B \in P$ then $C \in P$, where $C = \{xy \mid x \in A$ and $y \in B\}$. **Note:** You do not know where the concatenation break is.

3. P is closed under the star operation: if $A \in P$, then $B \in P$, where $B = \{x_1x_2\ldots x_n \mid n \geq 0$ and $x_i \in A\}$. **Hint:** Use dynamic programming.

Problem 3

Define the following two languages:

- **STRONGLYCONNECTED** = \{G \mid G is a directed graph such that there exist a path from $x$ to $y$ for any two vertices $x, y$\}

- **PATH** = \{(G, p, q) \mid G is a directed graph such that there exist a path from vertex $p$ to vertex $q$ in $G$\}

Given two languages $A$ and $B$, a function $f : \Sigma^* \to \Sigma^*$ is a mapping reduction if for all strings $x$, if $x \in A$ then $f(x) \in B$, and if $x \notin A$ then $f(x) \notin B$. 

---

1. [Diagram for Problem 2]

2. [Diagram for Problem 2]

3. [Diagram for Problem 2]

4. [Diagram for Problem 2]

5. [Diagram for Problem 2]

6. [Diagram for Problem 2]

7. [Diagram for Problem 2]

8. [Diagram for Problem 2]

9. [Diagram for Problem 2]

10. [Diagram for Problem 2]

11. [Diagram for Problem 2]

12. [Diagram for Problem 2]

13. [Diagram for Problem 2]

14. [Diagram for Problem 2]

15. [Diagram for Problem 2]
a. Give a polynomial time mapping reduction from Path to STRONGLYCONNECTED. Prove that your reduction is correct.

b. Is there a polynomial time mapping reduction from STRONGLYCONNECTED to Path? Justify your response.

You can assume that $G$ will have more than 2 nodes.

The following questions are lab problems.

Lab Problem 1

You are Qwop, our small nation’s sole representative at the Olympic Games. Ideally, you would like to run 100 meters, but our training program was under-funded. On the day of your event, you try to use the QWOP keys to move your legs. However, after a few laborious steps, you begin to fall. As you slowly descend to the ground, the following theoretical computer science problem enters your mind.

For this problem, only consider languages over the alphabet $\{0, 1\}$. For each of the following questions, give a proof if your answer is yes, else present a counterexample supporting your answer.

a. Let $L_1$ and $L_2$ be two languages in NP.
   (i) Must $L_1^*$ be in NP?
   (ii) Must $L_1 \cup L_2$ be in NP?
   (iii) Must $L_1 \cap L_2$ be in NP?

b. Now, suppose $L_1$ and $L_2$ are both NP-complete.
   (i) Must $L_1^*$ be NP-complete?
   (ii) Must $L_1 \cup L_2$ be NP-complete?
   (iii) Must $L_1 \cap L_2$ be NP-complete?

Lab Problem 2

Nyan Cat, Keyboard Cat, and Aristocat are having a kitty party and have a large collection of cat food they need to split. The cat food comes in tins and
bags, big and small, and in every variety you could ever think of. Always wanting to be fair, they want to make sure that everyone has the same amount of food, or else Keyboard Cat will get angry and play his horrible, repetitive music. In the end, the cats count that they have \( n \) packages of food. For each package \( i, 1 \leq i \leq n \), the item has a volume of \( v_i \), which is a positive integer written in binary.

\[
\text{FoodSharing} = \{ \langle v_1, \ldots, v_n \rangle \mid \text{each } v_i \text{ is a binary integer and there exists a way to split the set of integers into disjoint sets } S_1, S_2 \text{ and } S_3 \text{ such that the sum of the elements within each set } S_i \text{ is equal.} \}
\]

Prove that \( \text{FoodSharing} \) is NP-complete.

There are two components to the definition of NP-completeness: being NP-hard and being in NP. A language must satisfy both of these components to be NP-complete. Here is a series of steps you can follow:

1. Show that \( \text{FoodSharing} \) is in NP.
   
   (i) Construct an NTM \( M \) that decides \( \text{FoodSharing} \) in nondeterministic polynomial time, showing that \( \text{FoodSharing} \in \text{NP} \).
   
   (ii) Give a proof of \( M \)'s correctness. In particular, you must show that \( M \) accepts a string if and only if that string is in \( \text{FoodSharing} \).
   
   (iii) Give a proof that \( M \) runs in nondeterministic polynomial time.

2. Show that \( \text{FoodSharing} \) is NP-hard.
   
   (i) Construct a polynomial-time reduction from an NP-hard language \( A \) to \( \text{FoodSharing} \), showing that \( \text{FoodSharing} \) is NP-hard.
   
   (ii) Give a proof of the reduction’s correctness. All “yes” instances in \( A \) must map to “yes” instances in \( L \), and all “no” instances in \( A \) must map to “no” instances in \( L \).
   
   (iii) Give a proof that the reduction runs in polynomial time.

For this problem and future problem sets, proving that a language is NP-complete requires all of these components.

**Hint:** We will soon see that

\[
\text{SubsetSum} = \{ \langle S, t \rangle \mid S \text{ is a set of numbers with a subset that sums to } t, \text{ where } S \text{ and } t \text{ are both encoded in binary.} \}
\]

is NP-complete. Try reducing to \( \text{FoodSharing} \) from this language.
Lab Problem 3

In the 80’s, Nokia took its indestructible phones to the next level by releasing a model that could solve several NP-complete problems such as SUBSET-SUM and KNAPSACK in polynomial time! This phone, dubbed the “Unary Phone 1980” only had one input button, a “1” key, and contributed greatly to Nokia’s success in the 90’s and 00’s.

However, Nokia’s having a bit of a troubled time lately, and they are attempting to build a 2020 model of their unary phone that can solve all problems in NP in polynomial time. They believe that if the language of a unary phone is NP-complete, this will allow them to show that P = NP. Unfortunately, they haven’t the faintest idea where to start with this, and hence they approach CS 1010 students seeking their help. They provide you with the following information:

A language is called unary if every string in it is of the form 1^i for i ≥ 0. We want to show that if there exists a unary language that is NP-complete, then we have P = NP.

Suppose there exists such a language L that is NP-complete. As L is NP-complete, any problem in NP can be reduced to it in polynomial time. In particular, there exists a polynomial time reduction from SAT to L. Call this reduction f.

We will use this reduction f to design an algorithm that solves SAT. Your job is to analyze the correctness of the algorithm, its time complexity, and implications for P versus NP.
Indeed, given a boolean formula $\phi = \phi(x_1, \ldots, x_n)$ in the variables $x_i$, you can set up a binary tree as shown above by recursively assigning the last unassigned boolean variable. The leaves of the above tree would be all possible assignments of $\phi$. Thus, $\phi$ is satisfiable if and only if at least one of the leaves in the above tree evaluates to true.

The heuristic behind our algorithm is as follows: enumerating the tree above takes exponential time $O(2^n)$; instead, we use the aforementioned reduction $f$ to the NP-complete unary language $L$ to “prune” the binary tree as we build it.

Consider the following procedure:

1. The input is a boolean formula $\phi = \phi(x_1, \ldots, x_n)$.
2. Initialize an array $A$ with the given boolean formula, i.e. $A = \{ \phi(x_1, \ldots, x_n) \}$.
   At this point $A$ consists of a boolean formula with $n$ unassigned variables.
3. For $i = n$ down to 1:
   i. For each boolean formula $\varphi \in A$, delete $\varphi$ from $A$ and replace it with $[ \varphi(x_1, \ldots, x_{i-1}, x_i = 0), \varphi(x_1, \ldots, x_{i-1}, x_i = 1) ]$. At this point, $A$ consists of the boolean formula $\phi$ with $i - 1$ unassigned variables.
ii. Initialize an empty array $H$.

iii. For each boolean formula $\varphi \in A$, if $f(\varphi) \in 1^*$ and for no formula $\theta \in H$ does $f(\varphi) = f(\theta)$, then add $\varphi$ to $H$.

iv. Set $A = H$.

4. If any of the elements of $A$ evaluate True, then return True. Else return False.

Note that after the last iteration of the for loop, $A$ is simply a subset of the set of all possible assignments of $\phi$. That is, all the elements of $A$ are formulas that have all variables assigned. Look to the binary tree if this isn’t clear.

Answer the following questions with regard to the above algorithm:

a. Analyze Step 3 (i) in the above procedure. Specifically, given $\varphi(x_1, \ldots, x_n)$, why can we replace it with $\varphi(x_1, \ldots, x_n = 1)$ and $\varphi(x_1, \ldots, x_n = 0)$?

b. Analyze Step 3 (iii). That is, given two boolean formulas $\varphi_1, \varphi_2$ such that $f(\varphi_1) = f(\varphi_2)$, why do we only keep one of them in the array $A$?

*Hint:* Does $\varphi_1$ having a satisfiable assignment have any implications for the satisfiability of $\varphi_2$, given $f(\varphi_1) = f(\varphi_2)$?

c. Use the Parts b. and c. to briefly describe why the algorithm correctly determines whether a boolean formula has a satisfiable assignment.

d. Prove that the algorithm runs in polynomial time.

*Hint:* Let $|f(\phi)|$ denote the length of the string $f(\phi)$. As $f$ is a polynomial time reduction, there exist constants $c, d \in \mathbb{Z}$ such that $|f(\phi)| \leq c \cdot n^d$ where $n$ is the number of boolean variables of $\phi$. Then, for any formula $\phi'$ obtained by assigning any of the variables of $\phi$, we have $|f(\phi')| \leq |f(\phi)|$.

e. Conclude that if a unary NP-complete language such as $L$ were to exist, then $P = NP$. 