CSE 431 Winter 2025
Assignment #5

Reading assignment: Read sections 9.3 and 7.5.

Problems:

1. (20 points) Show that if $P = NP$ then every language $A \in P$, except $A = \emptyset$ and $A = \Sigma^*$, is NP-complete.

2. (20 points) Let $U = \{\langle M, x, 1^t \rangle \mid M$ is an NTM that accepts input x within t steps$\}$. Show that U is NP-complete.

3. (20 points) Let ϕ be a 3CNF-formula. An NAE assignment to the variables of ϕ is one that satisfies ϕ but does not set all three literals to true in any clause.

   (a) Show that the negation of an NAE assignment for ϕ is also an NAE assignment for ϕ.

   (b) Let *NAESAT* be the set of all 3CNF formulas ϕ that have an NAE assignment. Prove that *NAESAT* is NP-complete. For the hardness part use a reduction from 3SAT. (Hint: Use the function that replaces each clause C_i of ϕ of the form $(y_1 \vee y_2 \vee y_3)$ where y_1, y_2, y_3 are literals by the two clauses $(y_1 \vee y_2 \vee z_i)$ and $(\overline{z_i} \vee y_3 \vee w)$ where w is a single new variable for all clauses and there is one z_i variable per original clause.)

4. (20 points) Let $01ROOT = \{\langle p \rangle \mid p$ is a polynomial in n variables with integer coefficients such that $p(x_1, \ldots, x_n) = 0$ for some assignment $(x_1, \ldots, x_n) \in \{0,1\}^n\}$.

   (a) Show that $01ROOT \in NP$.

   (b) Show that $3SAT \leq_m^P 01ROOT$. (HINT: First figure out how to convert each clause into a polynomial that evaluates to 0 iff the clause is satisfied. Then create a polynomial q that evaluates to 0 if and only if all of its inputs are 0. Finally, figure out how to combine the individual polynomials for the clauses using the polynomial q.

5. (20 points) If you take product-of-sums (CNF) or sum-of-products (DNF) to represent some Boolean function on n bits, there are 2^n rows in the truth table (number of possible clauses or terms). For each row, there is a size n OR required in the usual product-of-sums CNF construction. This yields a circuit of size $O(n2^n)$ for any Boolean function on n bits. In this question you will see that you can do better.

   A *(Boolean) decision tree* defined on inputs $x \in \{0,1\}^n$ is a rooted binary tree (not necessarily complete) with each internal node labeled by some input variable x_i for $i \in \{1, \ldots, n\}$ and each leaf labeled by an *output value*, 0 or 1.

   We define the computation of a decision tree as follows:

- The decision tree computation starts at the root of the tree.
- At a node v labeled x_i, if the value of x_i is 0, the computation moves to the left child of v and if the value of x_i is 1 it moves to the right child of v.
- The output of the tree on input x is the label of the leaf reached by the computation.

A decision tree T computes a function $f : \{0,1\}^n \to \{0,1\}$ iff for all $x \in \{0,1\}^n$ the leaf label reached by x is $f(x)$.

(a) Show that any Boolean function on n bits can be computed by some decision tree with at most $O(2^n)$ nodes.

(b) Design a circuit whose graph is a tree (you get to write input bits in multiple places) that computes the following function on 3 bits

$$h(a,b,c) = \begin{cases} b & \text{if } a = 0 \\ c & \text{if } a = 1. \end{cases}$$

(c) Show that that every Boolean function can be computed in size $O(2^n)$ by showing if a Boolean function f can be computed by with S nodes, then there is a circuit C of size $O(S)$ computing f such that each internal gate of C is used as the input to at most one other gate. (That is, the circuit is a tree.)

A. (Extra Credit) In this problem you will save a factor of $O(n)$ in the worst-case circuit size for computing Boolean functions on n-bit inputs by using circuits that aren't trees and that get to re-use their computed values for more than one higher level gate.

(a) Given a circuit C and g a gate of G, we can define another circuit C^g where the gate g is now designated as the output gate. Produce a construction, re-using internal gate values, that inductively on t builds a circuit C_t defined on $\{0,1\}^t$, with the following properties:

  * C_t has size $O(2^{2^t})$ and exactly 2^{2^t} top-level gates G_t such that
  * for each Boolean function h on $\{0,1\}^t$ there is exactly one circuit C_t^g that computes h and has $g \in G_t$.

(b) In the construction from Problem 5, there can be many gates computing the same function of the input, particularly at the lower levels. Here, you will avoid repeating such circuits. Combine the construction from Problem 5 for the top $n - t$ levels and that of part (a) in this problem for the lowest t levels for $t = \log_2 n - 1$ (rounded down to the nearest integer) to show that for every Boolean function f on n bits there is a circuit of size $O(2^n/n)$ computing f.