CSE 421 Spring 2025: Set 8

Instructor: Chinmay Nirkhe
Due date: **Friday** June 6th, 2025 11:59pm

Instructions: Solutions should be legibly handwritten or typeset (ideally in LaTeX). Mathematically rigorous solutions are expected for all problems unless explicitly stated.

You are encouraged to collaborate on problems in small teams but everyone must individually submit solutions. Solutions for the problems may be found online or in textbooks – but do not use them.

For grading purposes, list, with each problem, the names of your collaborators. Please start each problem on a new page.

Problem 1 (Integer programming)**.** In this problem, we prove that a variant of linear programming is NP-complete. Specifically, we will restrict our focus to *boolean integer* programming where each the resulting solution must have each x_i as 0 or 1 (the constraints must also be integers, but not necessarily 0 or 1). Instead of maximizing an objective function, our goal is simply to determine whether there exists a solution in the feasible region.

Formally, given an integer matrix A and integer vector b, we want to write an algorithm that outputs:

$$\text{IP}(A, b) = \begin{cases} 1 & \text{if there is a length } n \text{ binary vector } x \text{ that satisfies } Ax \leq b \\ 0 & \text{otherwise.} \end{cases}$$

1. **[2 points]** Prove that IP is in NP. Hint: Given A, b, and a possible solution x, give a polynomial time algorithm that checks if x is a correct solution. (Our answer to this part was 1-2 sentences.)

2. **[8 points]**. Prove that Vertex Cover $\leq_p$ IP. I.e. Show that given an instance of Vertex Cover, how to efficiently convert it into an instance of IP. In other words, construct A and b from the Vertex Cover graph, and show a bijection between solutions to the Vertex Cover instance and solutions to the IP instance.

Problem 2 (Finding solutions)**. [10 points]** If $\mathsf{P} = \mathsf{NP}$, then every *decision problem* in NP can be solved in polynomial time. Show that then there is a poly-time algorithm for *finding* a valid solution to Decision-Knapsack.

Formally, prove that if $\mathsf{P} = \mathsf{NP}$, there exists a poly-time algorithm which outputs a valid Knapsack set $S \subseteq [n]$ for the problem $(W, V, w_1, \ldots, w_n, v_1, \ldots, v_n)$ if it exists and outputs $\perp$, if no valid Knapsack set exists.

Problem 3. In this problem we will explore the fickle nature of P and NP – we will see that a variant of 3-SAT will be NP-complete while a similar, but slightly different, variant is trivial. Such distinctions are important to observe and understand because they help elucidate the landscape of algorithmic complexity. Recall that a 3-CNF φ is the AND of clauses $\varphi_1, \varphi_2, \ldots, \varphi_m$ where each clause φ_i is the OR of at most 3 literals[1].

1. **[10 points]** Let $(3, \leq 3)$-SAT be the problem of deciding the satisfiability of a 3-CNF formula with *at most* 3 literals per clause, and at most 3 occurrences of any variable. Prove that $(3, \leq 3)$-SAT is NP-complete.

   Hint: For the reduction, start with a 3-SAT formula $\varphi(z)$ over n variables and replace a variable z_i which appears too many times with multiple variables $z_i^{(j)}$. Then add some clauses to ensure that the values assigned to the $z_i^{(j)}$ are consistent.

2. **[10 points]** Now let $(3, = 3)$-SAT be the problem of deciding the satisfiability of a 3-CNF formula with *exactly* 3 literals per clause, and at most 3 occurrences of any variable. Prove that every instance of $(3, = 3)$-SAT is always satisfiable (and, therefore, in P).

   Hint: Consider the bipartite graph with clauses on the left, and variables on the right. Connect a clause to a variable if the variable or its negation appears in the clause. Prove that this graph has a matching whose size is the same as the number of clauses. Use the matching to prover there exists a satisfying assignment.

3. **[2 bonus points]** Come up with an algorithm which *finds* the satisfying assignment for the previous part. You do not need to explain correctness of your algorithm.

[1] A literal is a variable or its negation.