CS 170 Homework 11

Due **Saturday 4/19/2025, at 10:00 pm (grace period until 11:59pm)**

1 Study Group (1 point)

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write "none".

2 NP Basics (8 points)

Assume A reduces to B in polynomial time. In each part you will be given a fact about one of the problems. What information can you derive of the other problem given each fact? Examples of information we can gather: "B is NP," "A is not in P," or "no additional information." Each part should be considered independent; i.e., you should not use the fact given in part (a) as part of your analysis of part (b).

For each part, briefly explain your reasoning.

(a) (2 points) A is in P.

(b) (2 points) B is in P.

(c) (2 points) A is NP-hard.

(d) (2 points) B is NP-hard.

3 Reduction Warm-ups (7 points)

Let us define the following problems below:

> **Undirected Hamiltonian Path:** we are given an undirected graph G as input, and want to determine if there exists a path in G that uses every vertex exactly once.
>
> **Longest Simple Path:** we are given a directed (not necessarily acyclic) graph G as input, and want to determine if there exists a path in the DAG that is of length k or more.
>
> **Longest Path in a DAG:** we are given a DAG and a variable k as input, and want to determine if there exists a path in the DAG that is of length k or more.

Both the Directed Hamiltonian Path problem and Longest Simple Path problem are known to be NP-complete. However, recall that the Longest Path in a DAG problem is in P.

Consider the following two incorrect proofs that the longest path in a DAG problem is NP-hard. For each proof, explain why the proof is incorrect.

(a) (3 points) In order to show that Longest Path in a DAG is NP-hard, we will reduce Longest Simple Path to Longest Path in a DAG.

Consider a DAG G on which we wish to determine whether there exists a path of length at least k. We construct an instance of Longest Simple Path as follows: use the same graph G and ask whether there is a simple path of length at least k in G.

Since G is a DAG, any path in G is automatically simple (because cycles are not possible). Therefore, the answer to the Longest Simple Path instance on G is yes if and only if the answer to the Longest Path in DAG instance is yes. Therefore, Longest Path in a DAG is NP-hard.

(b) (4 points) In order to show that longest path in a DAG is NP-hard, we will reduce Undirected Hamiltonian Path to Longest Path in a DAG.

Consider an undirected graph G on which we wish to see if a Hamiltonian Path exists. We will convert it to an instance of Longest Path in a DAG as follows. Use DFS to find a traversal of G and assign directions to all the edges in G based on this traversal. In other words, the edges will point in the same direction they were traversed and back edges will be omitted, giving us a DAG.

Then, this new directed graph will be our instance for Longest Path in a DAG – we need to check if its longest path has at least $|V| - 1$ edges. If so, then there must be a Hamiltonian path in G, as any simple path with at least $|V| - 1$ edges must visit every vertex.

4 Decision vs. Search vs. Optimization (8 points)

Recall that a vertex cover is a set of vertices in a graph such that every edge is adjacent to at least one vertex in this set.

The following are three formulations of the VERTEX COVER problem:

- As a *decision problem*: Given a graph G, return TRUE if it has a vertex cover of size at most b, and FALSE otherwise.
- As a *search problem*: Given a graph G, find a vertex cover of size at most b (that is, return the actual vertices), or report that none exists.
- As an *optimization problem*: Given a graph G, find a minimum vertex cover.

At first glance, it may seem that search should be harder than decision, and that optimization should be even harder. We will show that if any one can be solved in polynomial time, so can the others.

(a) (5 points) Suppose you are handed a black box that solves VERTEX COVER (DECISION) in polynomial time. Give an algorithm that solves VERTEX COVER (SEARCH) in polynomial time.

(b) (3 points) Similarly, suppose we know how to solve VERTEX COVER (SEARCH) in polynomial time. Give an algorithm that solves VERTEX COVER (OPTIMIZATION) in polynomial time.

5 Some Sums (10 points)

Given an array $A = [a_1, a_2, \ldots, a_n]$ of nonnegative integers, consider the following problems:

1 **Partition**: Determine whether there is a subset $S \subseteq [n]$ ($[n] := \{1, 2, \cdots, n\}$) such that $\sum_{i \in S} a_i = \sum_{j \in ([n] \setminus S)} a_j$. In other words, determine whether there is a way to partition A into two disjoint subsets such that the sum of the elements in each subset equal.

2 **Subset Sum**: Given some integer x, determine whether there is a subset $S \subseteq [n]$ such that $\sum_{i \in S} a_i = x$. In other words, determine whether there is a subset of A such that the sum of its elements is x.

3 **Knapsack**: Given some set of items each with weight w_i and value v_i, and fixed numbers W and V, determine whether there is some subset $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i \geq V$.

For each of the following clearly describe your reduction and justify its correctness.

(a) (5 points) Find a linear time reduction from Subset Sum to Partition.

(b) (5 points) Find a linear time reduction from Subset Sum to Knapsack.

6 Upper Bounds on Algorithms for NP Problems (9 points)

(a) (2 points) Recall the 3-SAT problem: we have n variables x_i and m clauses, where each clause is the OR of at most three literals (a literal is a variable or its negation). Our goal is to find an assignment of variables that satisfies all the clauses, or report that none exists.

Give a $O(2^n m)$-time algorithm for 3-SAT. Just the algorithm description is needed.

(b) (4 points) Using part (a) and the fact that 3-SAT is NP-hard, give a $O(2^{n^c})$-time algorithm for every problem in NP, where c is a constant (that can depend on the problem). Just the algorithm description and runtime analysis is needed.

Hint: Since 3-SAT is NP-hard, you can use it to solve any problem in NP!

Note: This result is known as $\mathsf{NP} \subseteq \mathsf{EXP}$.

(c) (3 points) Recall the halting problem from CS70: Given a program (as e.g. a `.py` file), determine if the program runs forever or eventually halts. Also, recall that there is no finite-time algorithm for the halting problem. Let us define the input size for the halting problem to be the number of characters used to write the program.

Let's now consider a "search" variant of the halting problem: given a program, we not only determine if the program runs forever, but if it halts, we also return its output.

Given an instance of 3-SAT with n variables and m clauses, we can write a size $O(n+m)$ program that outputs a valid assignment if the instance is satisfiable and runs forever otherwise. So there is a polynomial-time reduction from 3-SAT to the search variant of the halting problem.

Based on this reduction and part (b): Is the search variant of the halting problem NP-hard? Is it NP-complete? Justify your answer.

