The first page of your homework submission must be a cover sheet answering the following questions. Do not leave it until the last minute; it's fine to fill out the cover sheet before you have completely finished the assignment. Assignments submitted without a cover sheet, or with a cover sheet obviously dashed off without much thought at the last minute, will not be graded.

- How many hours would you estimate that you spent on this assignment?

- Explain (in one or two sentences) one thing you learned through doing this assignment.

- What is one thing you think you need to review or study more? What do you plan to do about it?

Question 1. Consider the following three variant problems:

- INDEPENDENT-SET: Given a graph G and an integer $k \geq 0$ as input, does G have an independent set of size k? (This is a so-called *decision problem*, which outputs a single bit, *i.e.* a True/False value.)
- MAX-INDEPENDENT-SET: Given a graph G as input, output the largest k such that G has an independent set of size k.
- FIND-MAX-INDEPENDENT-SET: Given a graph G as input, output a subset of vertices of G which is a maximum-size independent set.

Intuitively these are ordered from least to most informative: INDEPENDENT-SET outputs just a boolean; MAX-INDEPENDENT-SET outputs a value k; and FIND-MAX-INDEPENDENT-SET outputs an actual maximum independent set.

(a) Prove that

INDEPENDENT-SET $\leq_P$ MAX-INDEPENDENT-SET $\leq_P$ FIND-MAX-INDEPENDENT-SET.

(b) Prove that MAX-INDEPENDENT-SET $\leq_P$ INDEPENDENT-SET.

(c) Prove that FIND-MAX-INDEPENDENT-SET $\leq_P$ INDEPENDENT-SET.

Hint: you can use the INDEPENDENT-SET black box more than once, as long as you only use it a polynomial number of times.

We will often restrict ourselves to studying *decision problems* (which have a simple True/False answer) since they are much simpler to work with but tend to be "just as hard as" other variants which return more informative answers.

Question 2 (K&T 8.4)**.** Suppose you're consulting for a group that manages a high-performance real-time system in which asynchronous processes make use of shared resources. Thus the system has a set of *n processes* and a set of *m resources*. Each process specifies a set of resources that it requests to use. Each resource might be requested by many processes; but it can only be used by a single process. Your job is to allocate resources to processes that request them. If a process is allocated all the resources it requests, then it is *active*; otherwise it is *blocked*. You want to perform the allocation so that as many processes as possible are active. Thus we phrase the RESOURCE RESERVATION problem as follows: Given a set of processes and resources, the set of requested resources for each process, and a number k, is it possible to allocate resources to processes so that at least k processes will be active?

Consider the following list of problems. For each problem either give a polynomial-time algorithm or prove that the problem is NP-complete.

(a) The general RESOURCE RESERVATION problem defined above.

(b) The special case of the problem when $k = 2$.

Remember that to show a problem is NP-complete, you must show *two* things: that the problem is in NP, and that it is NP-hard. Remember that a problem X is NP-hard if there is some other NP-hard problem H with a polynomial reduction $H \leq_P X$. Be sure to get this the right way around! You may assume that SAT, 3-SAT, INDEPENDENT-SET, VERTEX-COVER, and SET-COVER are all NP-hard.

(c) The special case of the problem when there are two types of resources—say, people and equipment—and each process requires at most one resource of each type. (In other words, each process requires at most one specific person and one specific piece of equipment.)

(d) The special case of the problem when each resource is requested by at most two processes.

Question 3. The SUBSET SUM problem asks, given as inputs a set of integers $\{x_1, \ldots, x_n\}$ and a target sum S, is there a subset of the x_i whose sum is exactly S? It turns out that this problem is NP-complete.[1] On an activity in class, you actually wrote a dynamic programming algorithm to solve this problem. Explain why you do not win $1 million.

[1] This can be proved by reduction from 3-SAT, but you don't need to understand the proof to answer this question.