The goal of problem sets is to get you to explore concepts from class. This assignment must be submitted to GradeScope by October 24 at 3:00 PM. See the syllabus for more information.

For purposes of CompSci 260P, when proving a problem is \(\mathcal{NP}\)-complete, you do not need to prove it is in \(\mathcal{NP}\), even though the formal definition would require this. You need only give a reduction and an explanation for why it is correct; the most convenient way for the latter is by showing it produces no false positives and no false negatives. For the former, you may use any problem shown in class to be \(\mathcal{NP}\)-complete, and on an exam, you would be constrained to using such a problem.

1. The following is a version of the Independent Set Problem. You are given a graph \(G = (V, E)\) and an integer \(k\). For this problem, we will call a set \(I \subset V\) strongly independent if, for any two nodes \(v, u \in I\), the edge \((v, u)\) does not belong to \(E\), and there is also no path of two edges from \(u\) to \(v\); that is, there is no node \(w\) such that both \((u, w) \in E\) and \((w, v) \in E\). The **Strongly Independent Set** problem is to decide whether or not \(G\) has a strongly independent set of size at least \(k\).

In lecture, we saw that the regular **Independent Set** problem is \(\mathcal{NP}\)-complete. You can probably (correctly) guess that this variant is also \(\mathcal{NP}\)-complete. The key to that proof is to give a function that solves **Independent Set**, assuming that we have a working solution to **Strongly Independent Set** (but we cannot access the code for it). Write such a function. Your function’s running time must be polynomial time, not including the function call to **Strongly Independent Set**.

2. For this problem, please believe me when I say that **Subset Sum** and **Knapsack** are both \(\mathcal{NP}\)-complete. As of the time of this posting, we have not proven either in lecture. Suppose you want to prove that **Min-Cost Fast Path** is also \(\mathcal{NP}\)-complete. The key to that proof is to give a function that solves **Subset Sum** or **Knapsack**, assuming that we have a working solution to **Min-Cost Fast Path** (but we cannot access the code for it). Write such a function. Your function’s running time must be polynomial time, not including the function call to **Min-Cost Fast Path**. If you wish to redefine **Min-Cost Fast Path** to be a decision problem for this question, you may do so. The problem is defined in Problem Set 1. Note that the algorithm you gave when solving the problem did not have a polynomial running time.