Problem 1 (Matching Medical Students to Hospitals\textsuperscript{1}, 25 points). In this problem we’ll extend the deferred-acceptance algorithm to address some of the needs of the National Resident Matching Program, which has been used to assign graduating medical students to jobs as medical residents for over 60 years.

Suppose there are \( m \) hospitals, each with a certain number of available positions for hiring residents. There are \( n \) medical students graduating this year, each interested in a job at one of the hospitals. Each hospital has a ranking of students in order of preference, and each student has a ranking of hospitals in order of preference. We will assume there are more students graduating than the number of slots available in the \( m \) hospitals.

As before, our goal is to find a way to match each student to at most one hospital in such a way that all available jobs are filled (since there are more students than jobs, some students will not get a job). We also want the assignment to be \textit{stable}. In this setting, we will consider two types of instability.

1. There are students \( s \) and \( s' \) and a hospital \( h \) so that (a) \( s \) is assigned to \( h \), (b) \( s' \) is assigned to no hospital, and (c) \( h \) prefers \( s' \) to \( s \).

2. There are students \( s \) and \( s' \) and hospitals \( h \) and \( h' \) such that (a) \( s \) is assigned to \( h \), (b) \( s' \) is assigned to \( h' \), (c) \( h \) prefers \( s' \) to \( s \), and (d) \( s' \) prefers \( h \) to \( h' \).

An assignment is \textit{stable} if neither type of instability exists.

Show that there is always a stable assignment of students to hospitals and give an efficient algorithm to find one.

Problem 2 (Truthfulness in Stable Marriage\textsuperscript{2}, 10 points). For this problem, we will explore the

\textsuperscript{1}Based on Kleinberg-Tardos, Chapter 1, Problem 4.
\textsuperscript{2}Based on Kleinberg-Tardos, Chapter 1, Problem 8.
issue of truthfulness in the stable marriage problem. Can a man or a woman end up better off by lying about his or her preferences?

More concretely, we suppose each participant has a true preference order. Now consider a woman \(w\). Suppose \(w\) prefers man \(m\) to \(m'\), but both are low on her list of preferences. Can it be the case that by switching the order of \(m\) and \(m'\) on her list of preferences (i.e. by falsely claiming that she prefers \(m'\) to \(m\)) and running the men-proposing deferred-acceptance algorithm with this false preference list, \(w\) will end up with a man \(m''\) that she truly prefers to both \(m\) and \(m'\)? Resolve this question by doing one of the following two things. (a) Give a proof that for every set of preference lists, switching the order of a pair cannot improve a woman's partner. (b) Give an example of a set of preference lists for which there is a switch that would improve the partner of the woman who switched preferences.

Problem 3 (Stable Roommates, 10 points). The stable roommate problem is a variant on the stable marriage problem we considered in class in which any of the \(2n\) people can be matched (instead of matchings of one man and one woman, as in the stable marriage problem).

Specifically, our input is a set of \(2n\) people and each person's preference ranking of the other \(2n-1\) people. Our goal is to output a matching consisting of \(n\) pairs of people who will become roommates. Given a matching, a pair of people \((p_1, p_2)\) is unstable if both \(p_1\) and \(p_2\) would rather be paired to one another than to their assigned roommates. A matching \(\mu\) is stable if there are no unstable pairs.

For the stable marriage problem, we showed that a stable matching always exists. Is the same true for the stable roommate problem? Resolve this question by doing one of the following two things. (a) Give a proof that stable matchings always exist for the stable roommate problem. (b) Give an example of preference lists for which there is no stable matching.

Problem 4 (Making Change, 25 points). Suppose we have \(k\) types of coins, with \(v_1, v_2, \ldots, v_k\) and would like to make change for some amount \(W\) using the minimum number of coins. For example, for US coins, we have \(k = 4\) and \(v_1 = 1, v_2 = 5, v_3 = 10, v_4 = 25\). If \(W = 49\) then the optimal solution uses 7 coins (one quarter, two dimes, and four pennies). We may assume for simplicity that there is always a way to make exact change (i.e. do not worry about inputs like \(V = \{2\}\) and \(W = 3\)). Consider the following natural greedy algorithm:

Let \((V = \{v_1, \ldots, v_k\}, W)\) be the input and assume \(v_1 \leq v_2 \leq \cdots \leq v_k\)
\[ C \leftarrow \emptyset \]  // Start with an empty set of coins
While \(W > 0\)
  Let \(v_i\) be the largest coin value such that \(v_i \leq W\)
  Add a coin of value \(v_i\) to \(C\)
  \(W \leftarrow W - v_i\)
Output \(C\)

1. [20 points] Prove that for US coins with \(V = \{1, 5, 10, 25\}\) and every amount \(W\), the greedy algorithm returns a set \(C\) of minimum size.

2. [5 points] Prove that, for some coin values \(V\), and some amount \(W\), the greedy algorithm does not find the optimal solution.
Problem 5 (Minimum Subcovers, 30 points). Suppose we have a set $S = \{I_1, \ldots, I_n\}$ consisting of intervals $I_i = [a_i, b_i]$ for some $0 \leq a_i \leq b_i \leq 1$. We say that such a set covers the unit interval $[0, 1]$ if for every $x \in [0, 1]$, there is an interval $I \in S$ such that $x \in I$. If $C$ is a set of intervals that 1) covers $[0, 1]$ and 2) is a subset of $S$ then we say that $C$ is a subcover of $S$.

Design an efficient greedy algorithm that takes as input a set of $n$ intervals $S$, and returns the smallest set of intervals $C$ that is a subcover of $S$. That is, it returns a set $C \subseteq S$ that covers $[0, 1]$ and if $C' \subseteq S$ is a also a cover of $[0, 1]$, then $|C'| \geq |C|$. Specify the algorithm, prove that it returns a subcover of minimum size, and analyze its running time.\(^3\)

---

\(^3\)Hint: Any of the three strategies we saw for designing/analyzing greedy algorithms can be made to work here.