Reading: Chapters 7.8 - 7.12.

Question 1
Suppose you spend your next summer as a counselor at a summer camp to make a little dough. As a counselor, you may be asked to take on many roles over the course of the summer; life-guard, hike leader, science teacher (yeah, it’s a nerd camp), soccer coach, etc. However, to take on any given role, you must be qualified for that role. In particular, you must have taken a certain set of courses. For example, to be a life guard, you need to have taken a course in swimming and in first aid. A hike leader needs to have taken first aid as well as earth science, while a science teacher needs to have taken a course in earth science, physics, chemistry, and education.

The standard salary at this camp is pretty minimal, but it is augmented by the following “bonus” system. For each role that you are qualified to fill, you get paid an extra amount that depends on the role. For example, if you have taken courses in swimming, first aid, and earth science, you might get a bonus of $200 for qualifying as a life guard, and another $150 for qualifying as a hike leader. However, you would not qualify for the bonus associated with teaching science. To maximize your bonus, it seems obvious that you should take as many courses as possible. Unfortunately (as you know), taking courses isn’t free. Every course has a price (financially). So you are faced with the following dilemma: Which courses should you take to maximize your net profit (the difference between your bonuses and the costs of the courses you take)?

To make this problem more concrete, assume you are given a list of the n available courses C and the m potential roles R. Each course i ∈ C has a price p_i. Each role j ∈ R that you qualify for earns you a bonus b_j. To qualify for role j, you must have taken all the courses C_j ⊆ C. If S ⊆ C is a set of courses, and T ⊆ R is the set of roles that S qualifies you for, then the net profit of taking these courses is

\[ \sum_{j \in T} b_j - \sum_{i \in S} p_i. \]

Give an algorithm that uses this data to select a set of courses that maximizes your net profit.

Question 2
Consider the task of tiling a chessboard (8 × 8 squares) with dominoes which are the size of two squares. That is, placing dominoes on the board such that every domino covers exactly two squares of the chessboard and every square on the chessboard is covered by exactly one domino. Dominoes may be placed horizontally (1 × 2) or vertically (2 × 1). This is clearly not hard to do, and in fact, it is pretty clear that there are many many ways to do this.

Now suppose you remove two opposite corners from the chessboard. Can this new area still be tiled with dominoes? It turns out that the answer is “no.” After messing around with dominoes for a few
minutes you may begin to suspect that it can’t be done. But the easy way to see that it is impossible is to notice that the two opposite corner squares of a chessboard are always of the same color. If we consider the original black and white coloring on the chessboard, then the new area has 32 squares of one color and 30 squares of the other color. But every domino always covers exactly 1 white square and 1 black square. So any tiling will always leave at least 2 squares uncovered.

(a) One might ask whether this sort of problem is the only sort of problem preventing you from tiling a region. More precisely, is it the case that for any \( n \times n \) chess board where \( n \) is even, removing an equal number of white squares and black squares leaves a region that can be tiled with dominoes? Give a proof or a counterexample.

(b) Give an efficient algorithm that, given a region (a subset of squares from an \( n \times n \) chessboard), decides whether that region can be tiled with dominoes.

(c) \((\text{bonus})\) Suppose the answer for a particular input is “no.” How would you generate a simple-to-understand argument that would allow you to convince people that indeed, the board could not be tiled. Such an argument should be generated with the assistance of an algorithm, but the argument itself should not rely on either trusting your algorithm or understanding anything about networks or flows (the argument should be easy to verify).

**Question 3**

You have just discovered the Lost City of Atlantis. Apparently no one had thought to check Lake Cayuga. Inside the city lie untold riches, but getting at these riches will require quite a bit work; digging through rubble, scraping away years of seaweed, disarming booby-traps, hauling treasure to the surface, etc. Your first thought is to get your friends to help, but then you realize that they’d probably want to share the spoils. Robots are out too, since they’d cost a fortune. The only reasonable option left is to use highly trained lobsters. Obviously.

You decide that in total, there are \( m \) types of work that need to be done; \( T_1, T_2, \ldots, T_m \). For each type of work \( T_i \), you calculate the number of different lobsters \( q_i \) you’ll need assigned to that task. You then head over to Wegmans, buy out the \( n \) lobsters they have in stock, and begin training them. Unfortunately, you soon discover that not all lobsters are created equal: they vary in the tasks they can learn, and the number of these that they can remember at a time. In particular, for each lobster \( \ell_j \), there is a set \( S_j \subseteq \{T_1, T_2, \ldots, T_m\} \) and an integer \( r_j \), indicating that lobster \( \ell_j \) can learn to do at most \( r_j \) tasks, and these must be selected from \( S_j \).

With your background in network flows, assigning lobsters to tasks seems fairly straightforward. However, you soon realize that there is another complication. While training some of your lobsters, you notice that others have begun to fight amongst themselves rather than doing what you’d trained them to do. You turn to your lobster guidebook, and learn that you actually have \( k \) different species of lobster in your lobster workforce. It seems that lobsters of the same species are very territorial with each other. You realize that in addition to satisfying the constraints you were already working with, you also need to ensure that no two lobsters of the same species are assigned to the same task. After some careful study, you have determined the species \( s_j \) of each lobster \( \ell_j \). Give an algorithm that determines whether the given set of lobsters can be trained to do the required set of tasks.