You are strongly encouraged to read Section 11.1 from the textbook, even though it was not covered explicitly in class. It is a nice example of an algorithm giving an approximation guarantee other than 2, and the lower bound techniques involved in proving it. Also, it may help you with some of this homework.

(1) [10 points]
   Problem 11.6 from the textbook.

(2) [10 points]
   Problem 11.12 from the textbook.

(3) [10 points]
   In class, we saw the fractional linear program for MINIMUM \textit{s-t} CUT. Here it is again:
   \[
   \text{Minimize } \sum_{e \in E} x_e \cdot c_e \\
   \text{subject to } \sum_{e \in P} x_e \geq 1 \text{ for all } s-t \text{ paths } P \\
   x_e \geq 0 \text{ for all edges } e
   \]
   In the integer program, all the \(x_e\) are restricted to be 0 or 1. In the fractional version, they may take values in between. (In an optimum solution, they will never be larger than 1.) At the time, we discussed how this LP can be solved in polynomial time (in the size of the graph), even though the LP has exponentially many constraints. Here, you are asked to give a rounding of LP solutions such that the rounded version is \textit{no more expensive} than the fractional version. Thus, this gives another (very inefficient) algorithm to compute a minimum \textit{s-t} cut in polynomial time. It also proves that this LP has an integrality gap of 1, i.e., that its best integral solution is as good as its best fractional solution.
   
   (Hint: Interpret the \(x_e\) values as edge lengths. Then, for each node \(v\), you can assign (and compute) a distance \(d(s, v)\) from \(s\). Prove that there must be a value \(r\) (which you can find in polynomial time) such that the cut separating \(\{v \mid d(s, v) \leq r\}\) from \(\{v \mid d(s, v) > r\}\) is an \textit{s-t} cut and has cost no more than the fractional LP-solution you started with.)

(4) [10 points]
   Now, let us look at the following generalization of the Minimum \textit{s-t} cut problem: you have a graph \(G = (V, E)\) with non-negative edge costs \(c_e\), as well as \(k\) terminals \(t_1, \ldots, t_k\). Your goal is to remove an edge set \(E' \subseteq E\) of minimum total cost \(\sum_{e \in E'} c_e\) such that \textit{each pair} of terminals is disconnected. This problem is NP-hard even for \(k = 3\). (You don’t need to prove this.)
   
   (a) Give an integer LP formulation of this problem. You are explicitly allowed to use exponentially many constraints, so long as you satisfy part (b).
   
   (b) Show that when you relax the ILP to an LP, it can be solved in polynomial time. If you used exponentially many constraints, you need to provide a membership and separation oracle.
   
   (c) Show that your LP has an integrality gap of (at least) 2.

(5) [10 points]
   You are given a universe \(U\) of \(n\) elements. For each \(i = 1, \ldots, m\), you are given two subsets \(S_i, T_i \subseteq U\). Your goal is to select exactly one of \(S_i, T_i\) for each \(i\), while minimizing the size of the union of the sets selected. As an application, imagine that you need to do \(m\) jobs, and job \(i\) can be performed either by team \(S_i\) or by team \(T_i\). Your goal is to hire as few people total as possible, while ensuring that all jobs can be performed with subsets of the teams you hired.
   
   Give (and analyze) a polynomial time 2-approximation algorithm for this problem. (Hint: formulate an LP and round it appropriately. Though other techniques may also work.)
(6) [0 points]

**Chocolate Problem (1 chocolate bar):**

Here, we explore an NP-complete variation on a flow problem. You are given a complete bipartite graph with \( n \) nodes on the left and \( m \) nodes on the right. All edges have infinite capacity. Each node \( v_i \) on the left has a supply of \( s_i \geq 0 \) units of flow (so its demand is \(-s_i\)). Each node \( u_j \) on the right has a demand of \( t_j \geq 0 \) units of flow. You may assume that \( \sum_i s_i = \sum_j t_j \), since otherwise, the problem is trivially not solvable.

Now, it is trivial that there is always a feasible circulation. However, our goal is to find a circulation using as few edges as possible. In other words, imagine that for every edge that you use, you have to pay a unit price, regardless of how much flow you send on it. You want to minimize the number of edges you need to buy in this way. This problem is NP-complete (a simple reduction from \textsc{Subset-Sum}, for instance). Our goal is to approximate it.

As a warmup, give and analyze a polynomial-time 2-approximation algorithm. As the “real” problem, give and analyze a polynomial-time \( \frac{5}{4} \) approximation algorithm.

(7) [0 points]

**Research Question:** The best approximation for the preceding problem that I know of is a \( \frac{5}{4} \) approximation. (It’s a not-too-difficult generalization of the previous problem, but requires knowing about the approximability of \textsc{Set Packing}.) Can you come up with a better approximation ratio than that? Presumably, your approximation factor won’t be better than \( \frac{177}{176} \), as approximating it better than that is NP-hard.