1 Graduation Requirements (33 points)

John Hopkins University has some nonstandard graduation requirements. Every time a student takes a class, they get some grade in $[0, 1]$. They must satisfy several requirements of the form “the sum of your grades in courses taken from set \( S \) must be at least \( k_i \).” The goal of this problem is to take on the role of the student, and figure out the least possible work they can do while still passing.

More formally, there is a set of \( n \) classes. Without loss of generality, we will simply say that this is the set \( [n] = \{1, 2, \ldots, n\} \). We are also given \( m \) subsets \( S_1, S_2, \ldots, S_m \) where each \( S_i \subseteq [n] \), and \( m \) values \( k_1, k_2, \ldots, k_m \in \mathbb{R} \). If a student puts in \( x_i \in [0, 1] \) amount of work into class \( i \), we assume that they will get \( x_i \) as a grade (i.e., the grade they receive is exactly equal to the amount of work they put into it). In order to graduate, for every \( i \in [m] \), the sum of the grades they receive in the classes in \( S_i \) must be at least \( k_i \) (not taking a class is equivalent to putting in no work, and hence getting a grade of 0). Our goal is to minimize the total amount of work the student has to do while still graduating.

Show how this problem can be solved in polynomial time by using linear programming. Be sure to specify what the variables are, what the constraints are, and what the objective function is.

2 Linear Programming for MSTs (34 points)

Let’s consider linear programming formulations of the minimum spanning tree problem. We now have an (undirected) graph \( G = (V, E) \) and weights \( w : E \to \mathbb{R}^+ \). As we discussed in class, one way of phrasing the minimum spanning tree problem is as finding the minimum cost connected subgraph which spans all nodes. This interpretation naturally gives rise to a straightforward LP relaxation which requires every cut to have at least one edge crossing it (fractionally). More formally, suppose that we have a variable \( x_e \) for every edge \( e \), and consider the following linear program. For all \( S \subset V \), let \( \delta(S) \) denote the edges with exactly one endpoint in \( S \) and exactly one endpoint not in \( S \).
\[
\min \sum_{e \in E} w(e)x_e \\
\text{subject to } \sum_{e \in \delta(S)} x_e \geq 1 \quad \forall S \subset V : S \neq \emptyset \\
x_e \geq 0 \quad \forall e \in E
\]

Note that there are an exponential number of constraints, but let's not worry about that.

(a) (17 points) Prove that the optimal value of this LP is at most the weight of the minimum spanning tree.

(b) (17 points) Unlike the shortest path LP, this is not an exact formulation of the MST problem. Find a graph \( G = (V, E) \) and weights \( w : E \rightarrow \mathbb{R}^+ \) such that the optimal LP value is strictly less than the weight of the MST (and prove this).

3 More Flows (33 points)

In class we (briefly) saw the Multicommodity Flow problem: given a directed graph \( G = (V, E) \), capacities \( c : E \rightarrow \mathbb{R}^+ \), and a collection of \( k \) source-sink pairs \( \{(s_i, t_i)\}_{i \in [k]} \), maximize the total flow sent (summed over all of the \( k \) commodities). In this setting each commodity \( i \) must itself be a valid \( s_i - t_i \) flow (satisfying flow-balance constraints), while together (summed over all commodities) they must satisfy all edge capacity constraints. We saw how to write a linear program for this problem.

Let's consider a different variant: the Scaled Multicommodity Flow problem. We are again given a directed graph \( G = (V, E) \), capacities \( c : E \rightarrow \mathbb{R}^+ \), and a collection of \( k \) source-sink pairs \( \{(s_i, t_i)\}_{i \in [k]} \), but we are also given demands \( d : [k] \rightarrow \mathbb{R}^+ \) for each commodity. Think of \( d(i) \) as the amount of commodity \( i \) that we want to send from \( s_i \) to \( t_i \). Of course, it might not be possible to satisfy all of this demand. In that case, we could just send as much as we can, which gets us back the Multicommodity Flow problem. But that might be very unfair to some commodities, who might (for example) get 0 flow while other commodities get their entire demand. To fix this, we will instead try to scale all demands down proportionally until we can actually satisfy this scaled demand.

Slightly more formally, in the Scaled Multicommodity Flow problem our objective is to find the largest value \( \lambda \) such that the it is possible to simultaneously send \( \lambda \cdot d(i) \) flow from \( s_i \) to \( t_i \) for each commodity \( i \in [k] \) subject to each commodity obeying the flow-balance constraints, and the total flow (summed over all commodities) satisfying the edge capacity constraints.

Show how to use linear programming to solve the scaled multicommodity flow problem. Be sure to specify what the variables are, what the constraints are, and what the objective function is.