Problem 1. (a) We can create a super-source S and a super-sink T such that S has an outgoing edge to all sources s_i and T has an incoming edge from all sinks t_i, and set the capacities of these auxiliary edges to be infinity. Then the max-flow from all the sources to all the edges is equal to the max-flow from the super-source to the super-sink.

(b) Building off the previous part, we can create a super-sink for the supply S with outgoing edges to all nodes n_i with non-zero supplies $s_i \neq 0$, where the capacity of each auxiliary edge is equal to the quantity of supplies s_i. Similarly, we can also create a super-sink node for the demand T with incoming edges from all nodes n_i with non-zero demands $d_i \neq 0$, where the capacity of each auxiliary edge is equal to the demand coming from that node d_i. Then, the question of whether there is a feasible flow that ships all the supply to meet all the demand reduces to whether the max-flow of the augmented graph from S to T is equal to the total amount of supply $\sum s_i = \sum d_i$.

(c) For each vertex $v \in V$, we can create an auxiliary vertex v' and route all incoming edges of v to v' instead without changing their capacities, i.e. (x, v) becomes (x, v') for all x that flow into v. Then, we create an edge from v' to v with capacity u_v. Thus, the total amount of flow through each vertex v is limited by the capacity of the corresponding edge $c(v', v) = u_v$, and all paths from x to v that exist in the original graph are still maintained in the augmented graph, thus satisfying the constraints of this variant.

Note: for each variant, we do not introduce any new computational steps to the max-flow algorithm besides some pre-processing to augment the graph which takes $O(n)$ time, so the overall time bounds for each variant do not change.

Problem 2. Consider a feasible flow f: it must satisfy the edge flow lower bounds constraint, i.e. $f \geq \sum_{(i,j)\in(S\times T)\cap E} l_{ij}$. Additionally, we know that the maximum reverse flow across the cut is upper-bounded by $\sum_{(i,j)\in(T\times S)\cap E} u_{ij}$. Thus, it follows that any feasible flow f satisfies:

$$f \geq \sum\nolimits_{(i,j)\in(S\times T)\cap E} l_{ij} - \sum\nolimits_{(i,j)\in(T\times S)\cap E} u_{ij}$$

Now, assume that the minimum value of all feasible flows $f_{min} > \sum_{(i,j)\in(S\times T)\cap E} l_{ij} - \sum_{(i,j)\in(T\times S)\cap E} u_{ij}$. This implies one of two scenarios: one, there is at least one edge across the cut for which the flow exceeds the lower bound, or two, there is at least one edge going across the reverse cut (i.e. from T to S) for which the flow has not yet reached the max capacity. In the first case, we can lower the flow from that edge, which implies that f is not the actual minimum feasible flow. In the second case, we know that the residual graph for the reverse flow from T to S has at least one augmenting path, so we can increase the reverse flow across that edge. In either case, it follows that any flow $f > \sum_{(i,j)\in(S\times T)\cap E} l_{ij} - \sum_{(i,j)\in(T\times S)\cap E} u_{ij}$ is not the minimum feasible flow, and therefore $f_{min} = \sum_{(i,j)\in(S\times T)\cap E} l_{ij} - \sum_{(i,j)\in(T\times S)\cap E} u_{ij}$, the maximum lower capacity bound.

Problem 3. We can construct a flow graph where one unit of flow is equivalent to one baseball game to determine whether the Red Sox will be eliminated from contention for the league pennant as follows:

- First, for each *pair* of teams that are not the Red Sox (e.g. teams i and j), we will create a node p_{ij} representing the number of games left to be played between these two teams; and accordingly, we will also construct a super-source S that has outgoing edges to each node p_{ij} with capacity $u(S, p_{ij}) = q_{ij}$ (the number of remaining games).
- Second, we will create a node t_i for each non-Red Sox team i, and for each pair node constructed previously, we will create two edges from that pair node to the respective team nodes, each with unlimited capacity, to represent the "outcome" of those matches; for instance, if team i wins 2 games against team j and loses 1, then $f(p_{ij}, t_i) = 2$ and $f(p_{ij}, t_j) = 1$.
- Third, we will create a super-sink node T with incoming edges from each of the team nodes t_i, where $u(t_i, T) = w_{RedSox} + \sum q_{i,RedSox} - w_i - 1$. This is because the maximum number of games that the Red Sox can win by the end of the season is their current number of wins plus all of their remaining games, so we want to limit the total wins of team i to be at most one less in order to prevent them from tieing with or surpassing the Red Sox.

Having constructed this graph, we can now reformulate the problem of whether the Red Sox can still win the league pennant into a max-flow problem: if the max-flow from S to T is equal to the total outgoing capacity of S, then this implies an outcome where all other teams play their remaining games out such that all teams end up with less games won than the Red Sox (assuming the Red Sox win all their remaining games), and thus it is still possible for the Red Sox to win the league pennant.

Problem 4. (a) Suppose there exists some reduced edge length that is negative, i.e. $l^d_{vw} < 0$. Then this would imply that $l^d_{vw} = l_{vw} + d_v - d_w < 0 \implies l_{vw} + d_v < d_w$, or in other words, there exists a path from source to w composed of the shortest path to v and the edge from v to w that is shorter than the shortest path to w, which is a contradiction. Therefore, we know that all reduced edge lengths must be non-negative.

(b) Suppose that the length of the shortest path to w in the original graph is d_w for some path $s \to v_1 \to v_2 \to \cdots \to v_k \to w$. Then under reduced lengths, the length of this path is $l^d_{sw} = l^d_{sv_1} + l^d_{v_1v_2} + \cdots + l^d_{v_kw}$. If we expand each term into its definition, the d_v's cancel out due to telescoping and we are left with $l^d_{sw} = l_{sv_1} + l_{v_1v_2} + \cdots + l_{v_kw} - d_w = 0$. From our result in part (a), we know that $l^d_{sw} \geq 0$, so this path is also a shortest path of the reduced lengths graph as well.

(c) Let the i-th highest bits length of an edge between v and w be denoted as l^i_{vw} and the corresponding reduced length path as $l^{i,d}_{vw}$, and the shortest path from source to w on the i-th iteration as d^i_w. Our scaling algorithm is as follows:

- Start with all edges of the graph set to 0. Shift in the highest-order bit of each edge so that each edge is 1 or 0. Solve the shortest paths problem using Dial's algorithm for the base case, then shift in the next bits and compute the reduced edge length using the previous shortest paths and run Dial's algorithm again to determine the new shortest paths. Repeat until all the bits have been processed, at which point the shortest paths have been determined for the full graph.
- Scaling up: denote the i-th highest bits length of an edge between v and w as l^i_{vw} and the corresponding reduced length path as $l^{i,d}_{vw}$, and the shortest path from source to w on the i-th iteration as d^i_w. Then the equation for scaling up is:

$$l^{i+1,d}_{vw} = l^{i+1}_{vw} + d^i_v - d^i_w$$

  Thus, we can calculate the new reduced edge lengths for the $i+1$th iteration using the $i+1$ bit-shifted edge lengths and the previous step's shortest paths.

Since there are a total of $O(\log C)$ iterations to scale up, and Dial's algorithm runs in $O(m+D)$ where D is the maximum distance to be scanned using the reduced edge lengths, our scaling algorithm runs in $O(m \log C)$ time.

(d) Scaling is conducted bit-wise, so using base-k scaling results in $O(\log_k C)$ iterations of the algorithm. Previously we noted that Dial's algorithm runs in $O(m + D)$ time, where D is the maximum distance to be scanned. However, D also scales with the number of iterations, and the maximum distance scanned on each iteration is bounded by nk. Thus, the overall runtime of Dial's algorithm is $O(m+nk)$, which means we can set $k = m/n$ to achieve a $O(m)$ runtime for each individual iteration, and improve our runtime to $O(m \log_{m/n} C)$ overall for our scaling algorithm.

Problem 5. (a) Given the original graph G, we can construct an auxiliary graph G' for the purposes of solving the min-flow problem as follows:

- For each edge $(i, j) \in G$ with capacity lower bound l_{ij} and upper bound u_{ij}, we will create a corresponding edge (i', j') in G' with capacity $u_{ij} - l_{ij}$, representing the variable flow through this edge in the original graph satisfying the edge flow constraints.
- To satisfy the min-flow conditions, we will construct a super-source S and a super-sink T. Then, for each edge $(i, j) \in G$, we will create two edges (S, i') and (j', T) in G', each with capacity l_{ij}, to represent the min-flow through this edge in the original graph.

Thus, we can apply any standard max-flow algorithm to determine the max-flow through the auxiliary graph G' from S to T. If the max-flow determined by the algorithm is equal to the total outgoing capacity from S, then this indicates that the lower bound capacity constraints are satisfied, and the determined flow through G' corresponds to a feasible min-flow for the original graph G. On the other hand, if the max-flow is not equal to the total outgoing capacity of S, then this indicates that there is no feasible flow through the original graph that satisfies the lower capacity bounds.

(b) Given any feasible flow f through the original min-flow graph G, we can construct a reverse-residual graph G'', where for each non-saturated edge in G with flow $f(i, j)$, we will create an edge in the reverse direction (j, i) in G'' with capacity $f(i, j) - l_{ij}$ to represent the amount of flow we can take away from f. Then, we can apply a max-flow algorithm to G'' to determine the maximum reverse flow, and subtract this reverse flow from the original flow f to obtain the minimum feasible flow of the original graph G. Combining this step with part (a), we can obtain any feasible flow f satisfying the min-flow constraints, and then optimize it into the minimum possible flow using two applications of the max-flow algorithm, thus solving min-flow as efficiently as max-flow.

(c) To model this problem as a min-flow problem, we can create a graph representation as follows: for each lecture i that lasts from a_i to b_i, we will construct two vertices a_i and b_i and create an edge (a_i, b_i) with lower capacity 1 and upper capacity ∞ to represent the attendance of that lecture, so that there are $2n$ vertices in the graph with n edges between the n start times and n end times. Then, we will create a super-source S with outgoing edges to every lecture start time, i.e. (S, a_i) for $i \in 1, 2 \ldots, n$, and a super-sink T with incoming edges from every lecture end time, i.e. (b_i, T) for $i \in 1, 2 \ldots, n$, representing the flow of students to lectures (where one unit equals one student). Each of these edges (S, a_i) and (b_i, T) will have 0 as the lower capacity bound and ∞ as the upper capacity bound, since multiple students may attend any lecture. Finally, for any lecture start time that occurs after another lecture's end time plus the time to commute between them (i.e. $b_i + r_{ij} \leq a_j$ for all pairs of lectures i and j), we'll also create an edge (b_i, a_j) with 0 lower/∞ upper capacity to represent students going from one lecture to another. Thus, with this graph, we can use min-flow to determine the minimum students needed to attend all the lectures, which is the sum of outgoing flow from the source S. Since each of the n edges corresponding to lecture attendance require at least one unit of flow, the min-flow through the graph will ensure that all lectures are attended by at least one student. The min-flow will also determine the minimum students needed because for each lecture, the flow will send students from previous lectures if possible before requiring another student to be sent from the source. Thus, this algorithm uses min-flow to solve the problem of determining the minimum number of students needed to cover all the lectures.