1 Graduation Requirements (66 points)

John Hopskins University\(^1\) has \(n\) courses. In order to graduate, a student must satisfy several requirements of the form “you must take at least \(k\) courses from subset \(S\)”. However, any given course cannot be used towards satisfying multiple requirements. For example, if one requirement says that you must take at least two courses from \(\{A, B, C\}\), and a second requirement states that you must take at least two courses from \(\{C, D, E\}\), then a student who has taken just \(\{B, C, D\}\) would not yet be able to graduate as \(C\) can only be used towards one of the requirements.

Your job is to give an efficient algorithm for the following problem: given a list of requirements \(r_1, r_2, \ldots, r_m\) (where each requirement \(r_i\) is of the form “you must take at least \(k_i\) courses from set \(S_i\)”), and given a list \(L\) of courses taken by some student, determine if that student can graduate.

(a) (22 points) Given the \(m\) requirements and list of \(L\) courses taken (as above), design a flow network so that the maximum flow is \(\sum_{i=1}^{m} k_i\) if and only if the student can graduate.

Solution. We will make a graph with two tiers. In the first tier \(V_1\), we will have one node for every course that the student has taken. In the second tier \(V_2\), we will have one node for every set of requirements. There will be three sets of edges. The source will have an edge of capacity 1 going to each node in \(V_1\). Every node \(r_i \in V_2\) will have an edge with capacity \(k_i\) going to the sink. And for every \(l \in V_1\) and \(r_i \in V_2\), there will be an edge from \(l\) to \(r_i\) iff \(l\) is one of the courses in the requirement \(r_i\). That edge will have capacity 1.

Formally, we will define a graph \(G = (V, E)\), where:

\[
\begin{align*}
V_1 &= \{a_l | l \in L\} \\
V_2 &= \{b_i | 1 \leq i \leq m\} \\
V &= V_1 \cup V_2 \cup \{s, t\} \\
E_0 &= \{(s, a_l) | a_l \in V_1\} \\
E_1 &= \{(a_l, b_i) | a_l \in V_1, b_i \in V_2, l \in r_i\} \\
E_2 &= \{(b_i, t) | b_i \in V_2\} \\
E &= E_0 \cup E_1 \cup E_2
\end{align*}
\]

\(^1\text{https://www.youtube.com/watch?v=JEH2ha1p0WA}\)
And for every edge \((u, v) \in E\), we will define a capacity function \(c\) such that:

\[
c(u, v) = \begin{cases} 
  k_i & \text{if } u \in V_2, v = t, \text{ and } u = b_i \\
  1 & \text{otherwise}
\end{cases}
\]

Figure 1 shows what this graph would look like for the example in the problem description. Intuitively, this graph captures the idea that each of the student’s courses can be used towards exactly one requirement. Each course receives at most one unit of flow from the source, which can then be passed on to exactly one of the course requirements (assuming integral flows). Then, each requirement \(r_i\) can pass on at most \(k_i\) flow to the sink. If all of the edges from the requirements to the sink are saturated, the course requirements will have been satisfied.

Now we claim that this network has a feasible flow of value \(\sum_{i=1}^{m} k_i\) iff there is some assignment of courses to requirements such that all the requirements are satisfied.

If direction: Suppose there is some assignment \(A = \{(l, r_i) | l \in L, i \in \{1, \ldots, m\}, l \in r_i\}\) of courses to requirements such that all the requirements are satisfied. That is, for all \(1 \leq i \leq m\), let \(A_i = \{l \in L | (l, r_i) \in A\}\), and suppose that \(|A_i| = k_i\). (If \(|A_i| > k_i\), i.e. the student has taken extra courses for that requirement, then without loss of generality, we can just use an arbitrary subset of \(A_i\) with size \(k_i\).)

We can use \(A\) to construct a flow \(f\) in \(G\) such that \(|f| = \sum_{i=1}^{m} k_i\). For every \((l, r_i) \in A\), we can set \(f(s, a_l) = 1\) and \(f(a_l, b_i) = 1\). Then we can set \(f(r_i, t) = k_i\). It should be obvious that this flow satisfies the capacity constraints. To see why it satisfies flow conservation, observe that the \(A_i\)s are disjoint; that is, each \(l \in L\) is used to satisfy at most 1 requirement. So if we set \(f(s, a_l) = 1\) and \(f(a_l, b_i) = 1\), we will not set \(f(a_l, b_j) = 1\) for any \(j \neq i\). This means that, if \(l \in A_i\) for some \(i\), then we will have 1 unit of flow going into \(a_l\) from the source, and 1 unit of flow leaving it. And for the requirement nodes, since \(|A_i| = k_i\), we will have set \(f(a_l, b_i) = 1\) for exactly \(k_i\) nodes \(a_l\), and so flow conservation is satisfied there too.

So we have shown that, if the course requirements are satisfied, there exists some flow \(f\) in \(G\) with \(|f| = \sum_{i=1}^{m} k_i\). To see that this is a max flow, consider the cut \((V - \{t\}, \{t\})\). The edges that cross this cut are all the ones from the \(b_i\)s to \(t\). The sum of their weights is \(\sum_{i=1}^{m} k_i\). Therefore, by the min-cut max-flow theorem, this flow must be maximum.

Only if direction: Suppose the graph \(G\) has a max flow \(f\) with \(|f| = \sum_{i=1}^{m} k_i\). Then there must be that much flow going into \(t\), which means that each \(b_i \in V_2\) must contribute exactly
Suppose \( a_i \in V_1 \) contributes 1 unit of flow to \( b_i \in V_2 \). Then \( a_i \) cannot send flow to any other node, since it can receive only 1 unit of flow from the source. This means that each \( a_i \in V_1 \) that sends flow to some \( b_i \in V_2 \) can be uniquely paired with that \( b_i \). We can use this to construct a set of course requirements \( A = \{(l, r_i) | f(a_i, b_i) = 1\} \). This completes our proof.

(b) (22 points) Using the previous part, design an algorithm for the problem which runs in \( O(|L|^2m) \) time. Prove correctness and running time.

**Solution.** We use the following algorithm, based on the flow network \( G \) we designed in the previous part.

\[
\text{graduation_requirements}(L, r_1, r_2, \ldots, r_m):
\]

- **construct the graph** \( G \)
- \( mf = \text{ford_fulkerson}(G) \)
- if \( mf = \sum_{i=1}^{m} k_i \):
  - return True
- else:
  - return False

We know that Ford-Fulkerson returns a maximum flow, so combined with the previous part this implies that the algorithm is correct. Thus it just remains to prove its running time. To see that it runs in \( O(|L|^2m) \), observe that \( |V| = |L| + m + 2 \), and \( |E| \leq |L| + |L|m + m \). (There are \( |L| \) edges going from the source to the nodes in \( V_1 \), and at most \(|L|m \) edges between \( V_1 \) and \( V_2 \) (if every course can be used for every requirement), and \( m \) edges going from \( V_2 \) to \( t \).) This is \( O(|L|m) \), and so it takes \( O(|L|m) \) time to construct the graph.

Ford-Fulkerson runs in time \( O(F|E|) \), where \( F \) is the value of the max flow. For our graph, the max flow will be at most \( |L| \) (to see why, consider the cut \((\{s\}, V-\{s\})\)). Since \( |E| = O(|L|m) \), this means that Ford-Fulkerson runs in \( O(|L|^2m) \) for this graph.

Now suppose that John Hopkins University changes their 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 graduating.

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 \( S_j \subseteq [n] \), 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 \( j \in [m] \), the sum of the grades they receive in the classes in \( S_j \) must be at least \( k_j \) (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.

Note that unlike the previous requirements, now if some class \( i \) appears in both \( S_j \) and \( S_{j'} \), then it will count towards both requirement \( j \) and requirement \( j' \).
(c) (22 points) 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.

**Solution:** Consider the following LP. There is a variable \( x_i \) for each class \( i \), representing the student’s work.

\[
\begin{align*}
\min & \quad \sum_{i \in [n]} x_i \\
\text{subject to } & \quad \sum_{i \in S_j} x_i \geq k_j \quad \forall j \in [m] \\
& \quad 0 \leq x_i \leq 1 \quad \forall i \in [n]
\end{align*}
\]

This LP has polynomial size of the original problem, so it can be solved in polynomial time. Let \( x^* \) be the optimal solution to LP, we will show that \( \sum_{i \in [n]} x^*_i \) is the optimal solution for the graduation requirements problem, by showing that there is a bijection between the solution of the original problem and the solution of the LP, with the same objective value.

While doing the identical mapping between the solutions, we can see that the objective function is clearly the same, because the total amount of work for solution \( x \) is \( \sum_{i \in [n]} x_i \).

If \( x \) is a valid solution to the original problem, then it is a valid solution to the LP, because the second constraint is satisfied by \( x_i \in [0, 1] \) in the problem description, and the first constraint is satisfied since the sum of the grades in the classes in \( S_i \) must be at least \( k_i \) for every \( i \in [m] \).

If \( x \) is a valid solution to the LP, then it is a valid solution to the original problem, because \( x_i \in [0, 1] \) by the second constraint, and the sum of the grades in the classes in \( S_i \) is at least \( k_i \) for every \( i \in [m] \) by the first constraint.

Therefore solving this LP (in polynomial time) solves the problem.

2 More Flows (34 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 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.

**Solution.** Consider the following LP. There is a variable $f^{(i)}_{u,v}$ for each $(u,v) \in E$ and $i \in [k]$, representing the flow from $u$ to $v$ for commodity $i$. There is also a variable $\lambda$. Our LP is

\[
\begin{align*}
\text{max} & \quad \lambda \\
\text{subject to} & \quad \sum_{u:(s_i,u) \in E} f^{(i)}_{s_i,u} - \sum_{u:(u,s_i) \in E} f^{(i)}_{u,s_i} \geq \lambda d(i) & \quad \forall i \in [k] \\
& \quad \sum_{u:(u,v) \in E} f^{(i)}_{u,v} = \sum_{u:(v,u) \in E} f^{(i)}_{v,u} & \quad \forall v \in V \setminus \{s_i, t_i\} \\
& \quad \sum_{i \in [k]} f^{(i)}_{u,v} \leq c(u,v) & \quad \forall (u,v) \in E \\
& \quad f^{(i)}_{u,v} \geq 0 & \quad \forall (u,v) \in E, \forall i \in [k]
\end{align*}
\]

In other words, this is the multicommodity flow LP but instead of maximizing the total flow, we maximize $\lambda$ subject to all commodities sending at least $\lambda$ fraction of their demand.

This LP has polynomial size of the original problem, so it can be solved in polynomial time. Let $(f^*, \lambda^*)$ be the optimal solution to LP, we will show that $\lambda^*$ is the optimal solution for the scaled multicommodity flow problem, by showing that there is a bijection between the solution of the original problem and the solution of the LP, with the same objective value.

Given a scaled multicommodity flow $F$ with scaling factor $\lambda$, let $f^{(i)}_{s_i,u}$ be the flow from $u$ to $v$ for commodity $i$, then the first constraint is satisfied because the flow going out of $s_i$ is at least $\lambda d(i)$. The second constraint is satisfied because the flow should be even out for each node except the source and the sink. The third constraint is satisfied because of the capacity of each edge, and the forth constraint is satisfied because the flow is non-negative.

Similarly, given a solution $(f, \lambda)$ of the LP, we can set up a flow $F$ where $f^{(i)}_{s_i,u}$ is the flow from $u$ to $v$ for commodity $i$. Then $F$ is a non-negative flow because of the second and the forth constraint. $F$ satisfies the capacity constraint because the third constraint. Thus $F$ is a valid solution to the original problem. $F$ also has scaling factor $\lambda$ because of the first constraint.

Therefore by solving this LP (in polynomial time) we can solve the problem.