Describe an efficient algorithm to find the best minimum \((s, t)\)-cut. You may assume the capacities are integers.

**Solution:** We’ll add a tiny bit of additional capacity to each edge. That way the capacity of each cut will increase by the number of edges in that cut multiplied by the increase in capacities. However, the number we add will be so small that any \(s, t\)-cut that isn’t a minimum cut does not become a minimum cut, no matter how many edges it contains.

Let \(G = (V, E)\). Let \(c' : E \to \mathbb{R}_{\geq 0}\) be a new capacity function where \(c'(e) = c(e) + 1/(|E| + 1)\). We compute and return a minimum capacity \(s, t\)-cut according to capacity function \(c'\). Creating the new capacities takes \(O(E)\) time, and we can use Orlin’s \(O(VE)\) time algorithm to compute the cut, so the total running time is \(O(VE)\).

The capacity of any \(s, t\)-cut \((S, T)\) with \(k\) edges crossing from \(S\) to \(T\) increases by exactly \(k/(|E| + 1) < 1\) when using the new capacity function \(c'\). Any \(s, t\)-cut that is not a minimum \(s, t\)-cut has capacity at least one greater than the minimum capacity \(s, t\)-cut. Therefore, the capacity of any minimum \(s, t\)-cut will not grow greater than the capacity of any cut that is not a minimum \(s, t\)-cut. However, a best minimum \(s, t\)-cut will see the smallest increase in capacity among all \(s, t\)-cuts. We conclude the algorithm will return a best minimum \(s, t\)-cut.

**Rubric:** 10 points total: 5 points for the algorithm; 3 points for justification; 2 points for running time analysis.
Describe and analyze an algorithm to choose a subset of UTD faculty to staff the ice cream flavor committee or correctly report that no valid committee is possible.

**Solution:** We create a flow network $H = (W, F)$ with capacity function $c : F \to \mathbb{R}_{\geq 0}$ to help solve the problem. We add one vertex $d_i$ to $H$ for each department $i$. We add one vertex $f_j$ to $H$ for each faculty member $j$. We add three vertices $a$, $b$, and $c$ representing faculty ranks assistant, associate, and full, respectively. Finally we add vertices $s$ and $t$ to $H$.

We add an edge $s \rightarrow d_i$ of capacity 1 for each department $i$ (each department is represented by one faculty member). We add an edge $d_i \rightarrow f_j$ of capacity 1 for each department $i$ and faculty member $j$ such that faculty member $j$ has an appointment in department $i$ (departments can only be represented by their own faculty members). For each faculty member $j$, we add an edge $f_j \rightarrow r$ of capacity 1 where $r \in \{a, b, c\}$ is the rank of faculty member $j$ (each faculty has a single rank and represents at most one department). Finally, we add an edge $r \rightarrow t$ of capacity $k/3$ for each $r \in \{a, b, c\}$ (there are $k/3$ faculty members of each rank). Now, we run the Ford-Fulkerson maximum flow algorithm to compute a maximum value $(s, t)$-flow in $H$.

Suppose there is a flow of value $k$. The capacities are integers, so the flow values will be integers. We iteratively find paths from $s$ to $t$ along edges with non-zero flow value. For each path $s \rightarrow d_i \rightarrow f_j \rightarrow r \rightarrow t$ we find, we represent department $i$ using faculty member $j$ of rank $r$. We then reduce the flow values on each edge of the path by one to finish an iteration. Indeed, the capacity constraints guarantee each department is represented once, each faculty member is used once, and each rank is represented $k/3$ times.

On the other hand, suppose there is a way to build a committee. There is a flow of value $k$ that can be constructed as follows: For each department $j$ represented by faculty member $i$ of rank $r$, we send one unit of flow along the path $s \rightarrow d_i \rightarrow f_j \rightarrow r \rightarrow t$. The capacities won’t be exceeded, because each department is represented once, each faculty members represents one department, and each rank is represented exactly $k/3$ times. By the contrapositive, we should report there is no assignment if there is no flow of value $k$.

There are $O(nk)$ edges in $H$. There is an $(s, t)$-cut of capacity $k$ with $s$ alone on one side, so the value of the maximum flow is at most $k$. Building $H$, running Ford-Fulkerson, and extracting the flow paths takes $O(nk^2)$ time total. Observe we could have used Orlin’s algorithm, but there are $O(n + k) = O(n)$ vertices, leading to a running time of $O(n^2k)$ instead.

**Rubric:** 10 points total: 5 points for the algorithm; 3 points for justification; 2 points for running time analysis. Running time must be given in terms of $n$ and $k$. A correct algorithm running in $O(n^2k)$ time is worth full credit.
Suppose you are given a magic black box that can determine in polynomial time, given an arbitrary graph $G$, the number of vertices in the largest complete subgraph of $G$. Describe and analyze a polynomial time algorithm that computes, given an arbitrary graph $G$, a complete subgraph of $G$ of maximum size, using this magic block box as a subroutine.

**Solution:** The following algorithm $\text{MagicMaxClique}(G)$ takes as input a copy of $G$ and returns a complete subgraph of $G$ of maximum size.

$$
\text{MagicMaxClique}(G):
\text{arbitrarily number the vertices 1 through } V
\text{for each vertex } v \in \{1, \ldots, V\}
\text{with } \leftarrow \text{size of largest complete subgraph in } G
\text{without } \leftarrow \text{size of largest complete subgraph in } G - v
\text{if with = without}
\text{remove } v \text{ from } G
\text{return } G
$$

The algorithm loops through the vertices. If the maximum complete subgraph size remains the same with or without a vertex $v$, then there are complete subgraphs of maximum size that don’t include $v$. We are safe to remove $v$ from the graph in this case and the algorithm will continue on to find a maximum size complete subgraph inductively. Otherwise, every maximum size complete subgraph contains $v$ and we must not remove it. Again, the algorithm finds a maximum size complete subgraph after that point inductively.

The algorithm performs $O(V)$ calls to the magic black box and spends $O(V + E)$ time otherwise, removing vertices. A polynomial number of calls to a polynomial time algorithm takes polynomial time, so the algorithm runs in polynomial time.

**Rubric:** 10 points total: 5 points for the algorithm; 3 points for justification; 2 points for running time analysis.
Prove the following problems are NP-hard.

(a) Given an undirected graph $G$, does $G$ contain a simple path that visits all but 17 vertices?

**Solution:** We reduce from HAMILTONIANPATH in an undirected graph. Let $G$ be an undirected graph for which we wish to solve HAMILTONIANPATH. We may assume $G$ contains at least 2 vertices or we could solve the problem in constant time without doing a reduction. Create a graph $G'$ by adding 17 isolated vertices to $G$. Return TRUE if and only if $G'$ contains a simple path that visits all but 17 vertices. It takes $O(V + E)$ time to create $G'$ and there is one call to the problem’s hypothetical algorithm. Therefore, the reduction takes polynomial time.

Suppose $G$ has a Hamiltonian path that visits all of its vertices. Then this same path visits all but the 17 isolated vertices we added to create $G'$. The algorithm will return TRUE if $G$ has a Hamiltonian path.

Now suppose $G'$ has a simple path that visits all but 17 vertices. The path cannot use any of the new vertices we added to $G$ or it would contain exactly one vertex and miss the $\geq 18$ remaining vertices of $G'$. Therefore, the path visits exactly the vertices of $G$. The algorithm returns TRUE only if $G$ has a Hamiltonian path.

**Rubric:** 5 points total: 2 points for the reduction; 1 point each for both parts of the if and only if correctness proof; 1 point for arguing about polynomial time.

(b) Given an undirected graph $G$, can we color each vertex of $G$ with one of 19 colors so that every edge touches two different colors?

**Solution:** We reduce from 3COLOR. Let $G$ be an undirected graph for which we wish to solve 3COLOR. Create a graph $G'$ be adding a complete graph $K$ over 16 vertices to $G$ and adding edges between every vertex of $K$ and every vertex of $G$. Return TRUE if and only if $G'$ can be properly colored with 19 colors. It takes $O(V + E)$ time to create $G'$ and there is one call to the problem’s hypothetical algorithm. Therefore, the reduction takes polynomial time.

Suppose $G$ has a proper 3-coloring. Use those same colors in the copy of $G$ in $G'$ and use each one of the $19 - 3 = 16$ additional colors to color the vertices of $K$. Every edge to or in $K$ has a unique color as one of its endpoints. The edges of $G$ already have different colors on their endpoints, because we started with a proper 3-coloring of $G$. Therefore, we have a proper 19-coloring of $G'$, and the algorithm does return TRUE if $G$ has a proper 3-coloring.

Suppose $G'$ has a proper 19-coloring. Each vertex in $K$ gets its own color, because it is adjacent to all other vertices of $G'$. The remaining $19 - 16 = 3$ colors are given to vertices of $G$ for a proper 3-coloring of $G$, because every edge in $G$ is incident to vertices of two distinct colors. The algorithm returns TRUE only if $G$ has a proper 3-coloring.

**Rubric:** 5 points total: 2 points for the reduction; 1 point each for both parts of the if and only if correctness proof; 1 point for arguing about polynomial time.
The DominatingSet problem asks, given a graph $G$, for a dominating set of minimum size. Prove that this problem is NP-hard.

**Solution:** We reduce from MinVertexCover. Let $G = (V, E)$ be an undirected graph for which we wish to solve MinVertexCover. We create a graph $G' = (V', E')$ by adding vertices and edges to $G$ as follows. For each edge $uv \in E$, we add a new vertex $w_{uv}$ along with edges $uw_{uv}$ and $vw_{uv}$. We compute a dominating set $S$ of minimum size in $G'$. We then replace each vertex $w_{uv}$ in $S$ with $u$ (unless $u$ is already in $S$, in which case we just remove $w_{uv}$ from $S$). Finally, we return $S$. It takes $O(V + E)$ time to create $G'$ and modify the set $S$. Therefore, the reduction takes polynomial time.

We now argue that the algorithm returns a vertex cover of minimum size. Suppose there is a vertex cover $S$ of size $k$ in $G$. It covers every edge of $G$, so it contains or dominates both endpoints of every edge. Also, it dominates the extra vertex $w_{uv}$ that we added for each edge $uv$. The algorithm will find a dominating set of size at most $k$ for $G'$.

On the other hand, suppose it finds a dominating set of size $k$. Every vertex $w_{uv}$ was either in the dominating set or dominated by one of $u$ or $v$. This remains true even after replacing $w_{uv}$ with $u$. As each vertex $w_{uv}$ is still dominated after the replacements, each edge $uv$ is covered. The algorithm will actually return a vertex cover of size $k$.

**Rubric:** 10 points total: 4 points for the reduction; 2 point each for both parts of the if and only if correctness proof; 2 points for arguing about polynomial time.

A reduction from the decision version of any of the problems is worth full credit.