Let $G = (V, E)$ be a directed graph with \textit{integer} edge capacities $c : E \rightarrow \mathbb{Z}_{\geq 0}$. Suppose you have already computed a maximum flow $f^*$ in $G$.

(a) Describe and analyze an algorithm to update the maximum flow after the capacity of a single edge is increased by 1.

\textbf{Solution:} We will assume $f^*$ is integral. We compute the residual graph $G_f$, using the \textit{new} capacities. If $G_f$ contains an augmenting path from $s$ to $t$, then we augment $f^*$ by pushing one unit of flow along the path. That is the whole algorithm. Computing a single residual graph and possibly pushing flow along a single path takes $O(E)$ time total.

If there is no augmenting path, then $f^*$ is already maximum for the new capacities and the algorithm is correct to do nothing. Otherwise, the path has a bottleneck of at least 1, because capacities are integral. Any $(s, t)$-cut including the minimum $(s, t)$-cut can have its capacity increase by at most 1 after increasing the capacity of a single edge by 1. Therefore, the new maximum flow has value at most 1 more than $|f^*|$, meaning one augmentation of one unit of flow is enough to find a new maximum flow. ■

\textbf{Rubric:} 4 points total: 3 points for the algorithm and analysis; 1 point for the proof.

(b) Describe and analyze an algorithm to update the maximum flow after the capacity of a single edge is decreased by 1.

\textbf{Solution:} We will assume $f^*$ is integral. Let $u \rightarrow v$ be the edge whose capacity is decreasing, and let $c(u \rightarrow v)$ refer to the \textit{original} capacity of $u \rightarrow v$ before the decrease. We assume $u, v \notin \{s, t\}$. We can guarantee this without loss of generality by subdividing $u \rightarrow v$ to be a path $u \rightarrow u' \rightarrow v' \rightarrow v$, assigning $c(u \rightarrow u') = c(u' \rightarrow v') = c(v' \rightarrow v)$ and $f^*(u \rightarrow u') = f^*(u' \rightarrow v') = f^*(v' \rightarrow v)$, and decreasing the capacity of $u' \rightarrow v'$ instead.

Now, if $f^*(u \rightarrow v) < c(u \rightarrow v)$, then the algorithm terminates. Otherwise, we decrement $f^*(u \rightarrow v)$ to create flow $f'$. The flow $f'$ does not satisfy the vertex conservation constraints for $u$ and $v$, because the flow incoming to $u$ is 1 unit too high and the flow entering $v$ is 1 unit too low.

Let $G' = G - (u \rightarrow v)$. We compute the residual graph $G'_{f'}$. If there exists an augmenting path from $u$ to $v$ in $G'_{f'}$, we push 1 unit of flow along that path, and update $f'$ accordingly. Otherwise, we push one unit along an augmenting path from $u$ to $s$ and one unit on an augmenting path from $t$ to $v$. The algorithm then terminates. Computing residual graphs and searching for paths all take $O(E)$ time, so the total running time is $O(E)$.

If $f^*(u \rightarrow v) < c(u \rightarrow v)$, then $f^*$ is still feasible after decreasing the capacity of $u \rightarrow v$, and it is still a maximum feasible flow. Otherwise, we need to correct the one unit of flow imbalance on $u$ and $v$ according to $f'$. If there is an augmenting path from $u$ to $v$ in $G'_{f'}$, then pushing along that path changes the flow imbalance on each of $u$ and $v$ by 1 unit so the new outgoing and incoming flows for each of $u$ and $v$ become equal. The new flow is feasible, and it still has the same value as $f^*$ so it must be a maximum flow.

Consider the final case. Let $S \subset V$ be the set of vertices reachable from $u$ in $G'_{f'}$, and let $T = V \setminus S$. Vertex $u$ has more flow incoming than outgoing, and the only vertices with more
flow leaving then entering are $s$ and $v$. However, that incoming flow cannot be coming from $v$, because then there would be a path from $u$ back to $v$ in $G'_{f'}$, contradicting the current case. We can follow a path of positive flow edges from $s$ to $u$ in $G'$, so we can follow those edges backward in $G'_{f'}$ to see $s \in S$ and that we can push one unit from $u$ to $s$. We have $v \notin S$ by assumption, so $v \in T$. By similar logic to $s \in S$, we have $t \in T$, and we can push one unit from $t$ to $v$. Finally, these two augmenting paths don’t intersect since there are no residual graph edges from $S$ to $T$, so we don’t decrease the flow on any edge twice when pushing along those paths. We are saturating every edge from $S$ to $T$ in $G$ and avoiding every edge from $T$ to $S$, since we cannot go from vertices of $S$ to vertices of $T$ in $G'_{f'}$. Therefore, $(S, T)$ is a minimum $(s, t)$-cut whose capacity was just decreased by 1, meaning the value of the maximum flow must have decreased by 1. Flow $f'$ must be a maximum flow according to the new capacities.

\begin{rubric}
6 points total: 4 points for the algorithm; 1 point for the analysis; 1 point for the proof.
\end{rubric}
Suppose you are given an $n \times n$ grid, some of whose squares are colored black and the rest white. Describe and analyze an algorithm to determine whether tokens can be placed on the grid so that

- every token is on a white square;
- every row of the grid contains exactly one token; and
- every column of the grid contains exactly one token.

Your input is a two dimensional array $\text{IsWhite}[1 .. n][1 .. n]$ of booleans, indicating which squares are white. Your output is a single boolean.

**Solution:** We build a bipartite graph $G = (U \cup W, E)$ with one bipartite set $U = \{u_1, \ldots, u_n\}$ representing the rows and another bipartite set $W = \{w_1, \ldots, w_n\}$ representing the columns. For each pair $(i, j)$ where $\text{IsWhite}[i][j] = \text{True}$, we add an edge $u_iw_j$ to $G$. Finally, we compute a maximum matching in $G$ using the algorithm from lecture and return True if and only if the maximum matching has $n$ edges. Building the graph takes constant time per entry in $\text{IsWhite}$, so $O(n^2)$ time. Graph $G$ has $O(n)$ vertices and $O(n^2)$ edges, and it takes $O(VE)$ time to compute a maximum matching using the algorithm from lecture, so the total running time is $O(n^3)$.

To prove correctness, we need to show there is a legal placement of tokens if and only if the maximum matching in $G$ contains $n$ edges. Suppose there is a legal placement of tokens. Let $E' \subseteq E$ be the subset of edges where $u_iv_j \in E'$ if and only if $(i, j)$ gets a token. The tokens go only on white squares, so edge $u_iv_j \in E$. Also, no row or column gets more than one token, so $E'$ forms a matching. It takes $n$ tokens to put a token on each row, so the size of this matching is $n$.

Now, suppose there is a matching $E' \subseteq E$ of size $n$. We place a token on each position $(i, j)$ where $u_iv_j \in E'$. Since $u_iv_j \in E$, position $(i, j)$ must be white. Further, no vertex $u_i$ is incident to more than one member of $E'$ so no row gets more than one token. Similarly, no column gets more than one token. Finally, there are $n$ tokens being placed, so we must be giving every row and column exactly one token.

**Rubric:** 10 points total: 5 points for the algorithm; 2 points for the analysis, -1 point for not giving the running time in terms of $n$; 3 points for the proof.
Suppose there are $n$ candidate DJs and $g$ different musical genres available. Describe and analyze an efficient algorithm that either assigns a DJ and a genre to each of the $3k$ sets, or correctly reports that no such assignment is possible.

**Solution:** We compute $n$ by counting the DJs, and if $n < k$, then we report no assignment. Otherwise, we will reduce this problem to computing a maximum flow in a directed graph. We build the directed graph $G = (\{s, t\} \cup P \cup G \cup D, E)$ and capacity function $c : E \rightarrow \mathbb{Z}_{\geq 0}$ as follows. We build a set $P$ of people vertices where each $p_i \in P$ is a single candidate DJ. We build a set $G$ of genre vertices where each $g_j \in G$ is a single genre. We build a set $D = \{d_1, d_2, d_3\}$ of day vertices where each $d_i \in G$ is a single day. Finally, we add a source vertex $s$ and target vertex $t$ to $G$.

For each candidate DJ $p_i$, we add edge $s \to p_i$ to $G$ and set $c(s \to p_i) := 3$. For each genre $g_j$ that candidate DJ $p_i$ is willing to play, we add edge $p_i \to g_j$ to $G$ and set $c(p_i \to g_j) := \infty$. For each genre $g_j$ and day $d_t$, we add edge $g_j \to d_t$ to $G$ and set $c(g_j \to d_t) := 1$. Finally, for each day $d_t$, we add edge $d_t \to t$ to $G$ and set $c(d_t \to t) := k$.

We then compute a maximum $s, t$-flow $f^*$ in $G$ using the augmenting path algorithm of Ford and Fulkerson. Because the capacities are integral, the flow will be integral as well. If $|f^*| < 3k$, we report there is no legal assignment of DJs. Otherwise, observe that every path from $s$ to $t$ is of the form $s \to p_i \to g_j \to d_t \to t$. We iteratively find an $(s, t)$-path $s \to p_i \to g_j \to d_t \to t$ whose edges have positive flow value, assign DJ $p_i$ to play genre $g_j$ on day $d_t$, decrease all flow values along the path by 1, and repeat until there is no such path.

Counting candidate DJs, checking if $n < k$, and reporting there is no assignment in that case takes $O(n)$ time. If we build graph $G$, then it contains $O(n + g)$ vertices and $O(ng)$ edges. The $(s, t)$-cut $(V \setminus \{t\}, \{t\})$ has capacity $O(k)$, so we can compute a maximum flow in $O(Ek) = O(ngk)$ time. Reporting the $O(k)$ DJ assignments takes $O(k)$ time, so the total running time is $O(ngk)$.

If $n < k$, then we cannot fill all $3k$ slots without somebody DJing more than 3 times, so the algorithm is correct to give up in that case. Now, suppose the algorithm finds a flow of value $3k$. Each time it assigns a DJ to play a set, it decreases the value of the flow by 1, so it must do $3k$ assignments. Each assignment is based on a path $s \to p_i \to g_j \to d_t \to t$ where by construction DJ $p_i$ is willing to play genre $g_j$. We have $c(s \to p_i) = 3$, so we assign each DJ at most 3 times; $c(g_j \to d_t) = 1$, so we assign each genre to a day at most once; and $c(d_t \to t) = k$, so we assign at most $k$ sets per day. Since we assign $3k$ sets total, each day gets exactly $k$ sets.

Finally, we prove that if there is some legal assignment for the sets, then the algorithm will find a flow of value $3k$. Let $f$ be the flow where $f(s \to p_i)$ is equal to the number of times DJ $p_i$ is assigned, $f(p_i \to g_j)$ is equal to the number of times DJ $p_i$ plays genre $g_j$, $f(g_j \to d_t)$ is equal to the number of times genre $g_j$ is played on day $d_t$ (so 0 or 1), and $f(d_t \to t)$ is equal to $k$. The numbers sum up to make $f$ an $(s, t)$-flow, we have capacities high enough to guarantee this flow is feasible, and this flow has value $3k$. ■

**Rubric:** 5 points extra credit total: 3 points for the algorithm; 1 point for the analysis, -1/2 point for not giving the running time in terms of the input size; 1 point for the proof.