Let $G = (V, E, w)$ be a directed graph with weighted edges $w : E \to \mathbb{R}$; edge weights could be positive, negative, or zero. We’re going to design another algorithm for computing all-pairs shortest paths. For simplicity, you may assume $G$ is complete, meaning $E = V \times V$.

(a) Let $v$ be an arbitrary vertex of $G$. Describe an algorithm that constructs a directed graph $G' = (V', E', w')$ with edge weights $w' : E' \to \mathbb{R}$, where $V' = V \setminus \{v\}$, and the shortest-path distance between any two nodes in $G'$ is equal to the shortest-path distance between the same two nodes in $G$. The algorithm should run in $O(V^2)$ time.

Solution: We describe the procedure $\text{RemoveVertex}(V, E, w, v)$.

\begin{verbatim}
RemoveVertex(V, E, w, v):
    V' ← V \ {v}
    E' ← ∅
    for each $x ∈ V'$
        for each $y ∈ V'$
            add $x \to y$ to $E'$
            $w'(x \to y) ← \min\{w(x \to y), w(x \to v) + w(v \to y)\}$
    return $(V', E', w')$
\end{verbatim}

We will argue that the new weights preserve the shortest path distances by describing how to transform walks in $G$ to cheaper walks in $G'$ and vice versa. Then, one graph cannot have lower shortest path distances than the other. Consider a walk $p$ in $G$ between two vertices $s$ and $t$. If $p$ does not use $v$, then all its edges exist in $G'$, and their weights are only lower. If it does use $v$, then let it contain the subwalk $x \to v \to y$. The subwalk from $s$ to $x$, $x \to y$, and the subwalk from $y$ to $t$ together cost less than the total weight of $p$ in $G'$.

Now, let $p$ be a walk in $G'$. The same walk exists in $G$. For each edge $x \to y$ in $p$ with $w'(x \to y) < w(x \to y)$, we can replace $x \to y$ with a walk $x \to v \to y$ of cost $w'(x \to y)$. After the replacements, we get a new walk in $G$ of the same cost as $p$.

The running time is determined by the nested for loops over $V - 1$ vertices each, so it is $O(V^2)$.

\begin{center}
Rubric: 4 points total. 2.5 for the algorithm. 1 for the proof. 0.5 for the running time analysis.
\end{center}

(b) Now suppose we have already computed all shortest-path distances in $G'$. Describe an algorithm to compute the shortest-path distances in the original graph $G$ from $v$ to every other vertex, and from every vertex to $v$, all in $O(V^2)$ time.

Solution: We describe the procedure $\text{RestoreVertex}(V, E, w, v)$. We assume there is a global matrix $\text{dist} : V \times V \to \mathbb{R}$ that already stores the shortest-path distance $\text{dist}(x, y)$ for any pair of vertices $x$ and $y$ in $G'$.
The shortest path from \( v \) to any vertex \( z \) starts with some edge \( v \to y \) and continues with a shortest path from \( y \) to \( z \). The algorithm is just taking the minimum over the total length of all such paths. Similarly, the algorithm takes the minimum over the total lengths of all paths from some vertex \( x \) to \( y \) plus the edge weight \( w(y \to v) \).

We now have two doubly nested for loops over \( V - 1 \) items, so the running time is \( O(V^2) \).

---

(c) Combine parts (a) and (b) into another all-pairs shortest path algorithm. Your algorithm should run in \( O(V^3) \) time.

**Solution:** We describe the procedure \texttt{RECURSIVEAPSP}(\( V, E, w, \text{\textit{dist}} \)).

\[
\texttt{RECURSIVEAPSP}(V, E, w): \\
\text{if } V \neq \emptyset \\
\text{Let } v \text{ be any vertex in } V \\
(V', E', w') \leftarrow \texttt{REMOVEVERTEX}(V, E, w, v) \\
\texttt{RECURSIVEAPSP}(V', E', w') \\
\texttt{RESTOREVERTEX}(V, E, w)
\]

If there are any vertices during a recursive call, then we remove one of them \( v \) using \texttt{REMOVEVERTEX}, preserving the other all-pairs shortest path distances. We recursively compute the smaller set of distances by induction and finally use those distances to get distances to and from \( v \).

Each recursive call involves at most \( V \) vertices, so all they each take \( O(V^2) \) time to run the two subroutines described in earlier parts. The recursion tree is a path, so there are \( V \) recursive calls. The total running time is \( O(V^3) \).

---

**Rubric:** 3 points total. 2 points for the algorithm. 0.5 points for the proof. 0.5 points for the running time analysis.
Consider the directed graph $G = (V, E)$ with non-negative capacities $c : E \to \mathbb{R}_{\geq 0}$ and an $(s, t)$-flow $f : E \to \mathbb{R}_{\geq 0}$ that is feasible with respect to $c$.

(a) Draw the residual graph $G_f = (V, E_f)$ for flow $f$. Be sure to label every edge of $G_f$ with its residual capacity.

(b) Describe an augmenting path $s = v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_r = t$ in $G_f$ by either drawing the path in your residual graph or listing the path's vertices in order.

Solution: There is an augmenting path $s \rightarrow a \rightarrow c \rightarrow d \rightarrow t$.  

Rubric: 2 points total.
(c) Let \( F = \min_i c_f(v_i \rightarrow v_{i+1}) \) and let \( f' : E \rightarrow \mathbb{R}_{\geq 0} \) be the flow obtained from \( f \) by pushing \( F \) units through your augmenting path. Draw a new copy of \( G \), and label its edges with the flow values for \( f' \).

\[
\begin{align*}
\text{The } (s,t)-\text{flow } f'.
\end{align*}
\]

Solution:

\textbf{Rubric:} 2 points total.

(d) Draw the residual graph \( G_{f'} = (V, E_{f'}) \) for flow \( f' \).

\[
\begin{align*}
\text{The residual graph } G_{f'}.
\end{align*}
\]

Solution:

\textbf{Rubric:} 1 point total.
(e) There shouldn’t be any augmenting paths in $G_{f'}$, implying $f'$ is a maximum flow. Draw or list the vertices in $S$ for some minimum $(s, t)$-cut $(S, T)$.

Solution: $S = \{s, a, b, c\}$, the set of vertices reachable from $s$.

Rubric: 2 points total.

(f) What is the value of the maximum flow/capacity of the minimum cut?

Solution: The value/capacity is 8.

Rubric: 1 point total.
Suppose we are given a flow network $G = (V, E)$ in which every edge has capacity 1, together with an integer $k \geq 0$. Describe an algorithm to identify $k$ edges in $G$ such that after deleting those $k$ edges, the value of the maximum $(s, t)$-flow in the remaining graph is as small as possible.

**Solution:** We begin by finding a minimum $(s, t)$-cut $(S, T)$ using Orlin’s algorithm. If there are fewer than $k$ edges going from $S$ to $T$, we return all of them (and however many more we need to each $k$ edges total). If there are more than $k$ edges going from $S$ to $T$, we return an arbitrary subset of $k$ of them.

Let $C = |S, T|$ be the capacity of the minimum $(s, t)$-cut. After removing the edges, the capacity of $(S, T)$ decreases to $\max\{C - k, 0\}$. Every cut has at least $C$ edges, so no cut can have its capacity decreased below $\max\{C - k, 0\}$. Therefore, removing the edges we chose decreases the capacity of the minimum cut as much as possible. By the max-flow min-cut theorem, it also decreases the value of the maximum flow as much as possible.

It takes $O(VE)$ time to run Orlin’s algorithm, and then we spend $O(E)$ time selecting edges to remove, so the whole algorithm takes $O(VE)$ time total.

**Rubric:** 10 points total: 5 for the algorithm, 3 points for the justifications, 2 for the running time analysis.