Let \( G = (V, E) \) be a directed graph with edge weights \( w : E \to \mathbb{R} \) (which may be positive, negative, or zero). Assume there are no negative length cycles in \( G \).

(a) Describe and analyze an algorithm that constructs a directed graph \( G' = (V', E') \) 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 two nodes in \( G \). The algorithm should run in \( O(V^2) \) time.

**Solution:** We construct \( G' \) using the following algorithm. For simplicity, we say \( w(u \to u) = 0 \), and \( w(u \to v) = \infty \) if \( u \neq v \) and \( u \to v \notin E \).

```plaintext
RemoveVertex(V, E, w, v):
V' ← ∅
for every vertex \( u \in V \setminus \{v\} \)
add vertex \( u \) to \( V' \)
E' ← ∅
for every vertex \( u_1 \in V' \)
for every vertex \( u_2 \in V' \)
add edge \( u_1 \to u_2 \) to \( E' \)
w'\((u_1 \to u_2)\) ← \min\{w(u_1 \to u_2), w(u_1 \to v) + w(v \to u_2)\}
return \((V', E', w')\)
```

We’re doing a constant amount of work for each of the \( O(V^2) \) edges in \( G' \), so the algorithm takes \( O(V^2) \) time.

Let \( P = u_1 \to \ldots \to u_k \) be any path in \( G \) where \( u_1 \neq v \) and \( u_k \neq v \). There is a path from \( u_1 \) to \( u_k \) in \( G' \) of weight at most \( w(P) \). For each pair of consecutive vertices \( u_i, u_j \) in \( P \) not equal to \( v \), we take the edge \( u_i \to u_j \) in \( G' \). If \( v \) lies between \( u_i \) and \( u_j \) in \( P \), then \( w'\((u_i \to u_j)\) \leq w(u_i \to v) + w(v \to u_j) \). Otherwise, \( w'(u_i \to u_j) \leq w(u_i \to u_j) \).

Similarly, let \( P' = u_1 \to \ldots \to u_k \) be any path in \( G' \). There is a path from \( u_1 \) to \( u_k \) in \( G \) of weight at most \( w'(P') \). For each pair of consecutive vertices \( u_i, u_j \) in \( P' \), we take the edge \( u_i \to u_j \) in \( G \) or the pair of edges \( u_i \to v \) and \( v \to u_j \), whichever is cheaper. We have \( w'(u_i \to u_j) \) equal to the total weight for the cheaper of those two options.

We conclude that shortest path distances are equal between vertices existing in the two graphs.

**Rubric:** 4 points total: 3 points for the algorithm and analysis; 1 point for the proof.

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

**Solution:** We use the following algorithm, assuming \( dist[u_1][u_2] \) is already set for all vertices \( u_1, u_2 \neq v \).
We have two for loops each doing a constant amount of work for each of the \(O(V^2)\) pairs of vertices in \(G\), so the algorithm takes \(O(V^2)\) time total.

Clearly, \(\text{dist}[v][v] = 0\). Now, let \(s \neq v\) be some source vertex. The shortest path from \(s\) to \(v\) ends with some edge \(u \rightarrow v\). The first for loop correctly considers all options for that last edge, summing the shortest path distance to \(u\) and the weight of that last edge along the potential path. At the end, it settles with the best option. Similarly, the algorithm considers all options for the first edge from a \(v\) to \(t\) shortest path, taking the best option.

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

(c) Finally, combine parts (a) and (b) to describe and analyze another all-pairs shortest path algorithm. This algorithm should run in \(O(V^3)\) time.

**Solution:** We use the following recursive algorithm. Here, an uninitialized array \(\text{dist}[1..n][1..n]\) is given as input to the top-level call of the algorithm.

```plaintext
RecursivelyAPSP(V, E, w, dist):
    if |V| = 0
        return
    else
        v ← some vertex
        (V', E', w') ← REMOVEVERTEX(V, E, w, v)
        RecursivelyAPSP(V', E', w', dist)
        DistancesWithVertex(V, E, w, v, dist)
```

The algorithm does \(O(V)\) recursive calls total, and the algorithm does \(O(V^2)\) non-recursive work in each recursive call. The total running time is \(O(V^3)\).

We can prove the algorithm is correct with a straightforward inductive proof. If \(|V| = 0\) there are no distances to compute and the algorithm is correct. Otherwise, we correctly remove some vertex \(v\) from the graph without effecting other shortest paths distances according to part (a), recursively compute these shortest path distances by induction on \(|V|\), and then correctly use those distances to figure out distances to and from \(v\) according to part (b).

\[\text{Rubric: 3 points total: 2 points for the algorithm and analysis; 1 point for the proof.}\]
Consider the directed graph $G = (V, E)$ given 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 $s = v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_r = t$ in $G_f$ by either drawing the path in your residual graph or listing the path's vertices in order.

Solution: The path $s \rightarrow b \rightarrow c \rightarrow t$ is an augmenting path. (So is $s \rightarrow a \rightarrow b \rightarrow c \rightarrow t$.)
(c) Let $F = \min c_f(v_i \rightarrow v_{i+1})$ 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 $f'$. Is your new flow a maximum flow in $G$?

**Solution:** Here’s the result of pushing along the first augmenting path mentioned above. Here, $F=5$.

![Diagram](image)

The new flow is a maximum flow, because it has value 10 and $(\{s, a, b\}, \{c, d, t\})$ is an $s,t$-cut with capacity 10. (If we drew the new residual graph, $\{s, a, b\}$ would be precisely the vertices reachable from $s$).

**Rubric:** 2 points total. -1 for not saying if the new flow is a maximum flow.

(d) Describe and analyze an algorithm to compute a graph $G' = (V', E')$ with non-negative edge capacities $c' : E' \rightarrow \mathbb{R}_{\geq 0}$ but no vertex limits so that the value of the maximum feasible flow in $G'$ with respect to $c'$ is equal to the value of the maximum feasible flow in $G$ with respect to both $c$ and $\ell$.

**Solution:** We’ll use the following algorithm.
We do a constant amount of work for every vertex and edge of $G$, so the algorithm runs in $O(V + E)$ time.

For correctness, observe that every feasible flow $f$ in $G$ can be converted to a feasible flow $f'$ in $G'$, by setting the flow on the $v_{in} \rightarrow v_{out}$ edges using $f'$ to be the amount of incoming flow to $v$ using $f$. The capacities of these intermediate edges are set equal to the vertices’ limits in $G$, so the incoming flow to $v_{in}$ will be at most the capacity of edge $v_{in} \rightarrow v_{out}$. Likewise, feasible flows $f'$ in $G'$ can be converted to feasible flows $f$ in $G$ by simply ignoring the flow values on the $v_{in} \rightarrow v_{out}$ edges. The flows maintain their values during these conversions. ■

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