Let $G = (V, E)$ be a directed graph with edge weights $w : E \to \mathbb{R}$ (which may be positive, negative, or zero), and let $s$ be an arbitrary vertex of $G$.

(a) Suppose every vertex $v$ stores a number $\text{dist}(v)$. Describe and analyze an algorithm that returns 'yes' if $\text{dist}(v)$ is the shortest-path distance from $s$ to $v$ for every vertex $v$. Otherwise, the algorithm should return 'no'.

Solution: Our algorithm is given in the following pseudocode.

```plaintext
CorrectDistances(V, E, w, s, dist):
    G' = (V, ∅)
    for every edge u→v
        if dist(u) + w(u→v) < dist(v)
            return 'no'
        else if dist(u) + w(u→v) = dist(v)
            add u→v to graph G'
    if dist(s) ≠ 0
        return 'no'
    mark vertices reachable from s in G'
    for every vertex v
        if v is unmarked and dist(v) ≠ ∞
            return 'no'
    return 'yes'
```

Looping over the edges at the beginning takes $O(E)$ time. Searching from $s$ takes $O(V + E)$ time. Therefore, the total running time is $O(V + E)$.

The algorithm correctly returns 'no' if an edge $u→v$ is tense (meaning $\text{dist}(u) + w(u→v) < \text{dist}(v)$) or if $\text{dist}(s) ≠ 0$. Now suppose neither case above holds. The algorithm builds a subgraph $G'$ containing edges $u→v$ for which $\text{dist}(u) + w(u→v) = \text{dist}(v)$. If a vertex $v$ is unreachable from $s$ in this subgraph, then there is no path from $s$ to $v$ for which the distances to vertices along the path correctly represent the length of the path to those vertices. This means either $v$ is not reachable from $s$ in $G$ or the distances are not set correctly to represent any paths, much less shortest paths. The algorithm correctly checks if any unreachable vertices do not have infinite distance from $s$ and correctly returns 'no' if that is the case.

Now, suppose the algorithm gets past those checks and returns 'yes'. For each vertex $v$, we have $\text{dist}(v)$ is the length of some path from $s$ to $v$ (or $\infty$ if there is no such path discovered). But we have established that no edge is tense, so these must all be shortest path distances.

(b) Suppose instead that every vertex $v ≠ s$ stores a pointer $\text{pred}(v)$ to another vertex in $G$. Describe and analyze an algorithm that returns 'yes' if these predecessor pointers define a single-source shortest path tree rooted at $s$. Otherwise, your algorithm should return 'no'.

Rubric: 5 points total: 3 points for the algorithm; 1 point for the proof; 1 point for the analysis.
**Solution:** Our algorithm is given in the following pseudocode.

```plaintext
CorrectPredecessors(V, E, w, s, pred):
    G′ = (V, ∅)
    for every vertex ν
        if pred(ν) ≠ NULL
            add edge pred(ν) → ν to G′
    attempt to topologically sort G′
    if G′ was not a DAG
        return 'no'
    for every vertex ν in topological order
        if ν = s
            dist(ν) ← 0
        else if pred(ν) = NULL
            dist(ν) ← ∞
        else
            dist(ν) ← dist(pred(ν)) + w(pred(ν) → ν)
    return CorrectDistances(V, E, w, s, dist)
```

The algorithm takes $O(V)$ time to build the subgraph $G'$. It then takes $O(V)$ time to topologically sort the subgraph and set the tentative distances. Finally, it takes $O(V + E)$ time to run CorrectDistances as given in part (a) for a total running time of $O(V + E)$.

The subgraph $G'$ should contain edges going back toward $s$ if the predecessor pointers define a tree, so the algorithm is correct to return ‘no’ if the pointers form a cycle. Finally, it sets the only distances possible if the predecessors form a shortest path tree: $dist(s) = 0$; $dist(ν) = ∞$ if $ν$ has no predecessor; and $dist(ν)$ equals the distance to its predecessor plus the weight of that last edge otherwise. The distances needed from the predecessors will always be available since we topologically sorted the vertices beforehand. The final call correctly checks if these distances defined by the proposed predecessor pointers actually make sense for shortest paths.

**Rubric:** 5 points total: 3 points for the algorithm; 1 point for the proof; 1 point for the analysis.
Let $G = (V, E)$ be a directed graph with edge weights $w : E \to \mathbb{R}$ (which may be positive, negative, or zero), and let $s$ be an arbitrary vertex of $G$.

(a) Suppose edge $u \to v$ is tense after $V$ phases of Shimbel’s single-source shortest path tree algorithm. What is the length of the shortest walk from $s$ to $v$?

**Solution:** Recall, after $i$ phases of Shimbel’s algorithm, $\text{dist}(v)$ is at most the length of the shortest walk from $s$ to $v$ consisting of at most $i$ edges. If $u \to v$ is still tense after $V$ phases, then the shortest walk using $V$ edges was still not the shortest walk from $s$ to $v$ possible. A simple path can only contain $V - 1$ edges, so these progressively shortest walks must be going around a cycle of negative length. A walk can go around the cycle an arbitrary number of times, so the length of the shortest walk from $s$ to $v$ is $-\infty$. $lacksquare$

**Rubric:** 3 points total, -2 for no explanation.

(b) Describe and analyze a modification of Shimbel’s algorithm that computes the correct shortest path distance from $s$ to every other vertex of $G$, even if the graph contains negative cycles. Specifically, if any walk from $s$ to $v$ contains a negative cycle, your algorithm should end with $\text{dist}(v) = -\infty$; otherwise, $\text{dist}(v)$ should contain the length of the shortest path from $s$ to $v$. The modified algorithm should still run in $O(VE)$ time.

**Solution:** We use the following modification to Shimbel’s algorithm. We expand the definition of tense and the relax step to make it more clear what our modifications are. We also remove the predecessor pointers, because a single pointer is meaningless when the shortest walk from $s$ must contain a cycle.

**SuperShimbelSSP($s$):**

```
INITSSP($s$)
repeat $V$ times:
    for every edge $u \to v$
        if $\text{dist}(u) + w(u \to v) < \text{dist}(v)$
            $\text{dist}(v) \leftarrow \text{dist}(u) + w(u \to v)$

repeat $V$ times:
    for every edge $u \to v$
        if $\text{dist}(u) + w(u \to v) < \text{dist}(v)$
            $\text{dist}(v) \leftarrow -\infty$
```

We are just looping over all the edges $2V$ times, so the total running time is $O(VE)$.

Again, after $i$ phases of Shimbel’s algorithm, $\text{dist}(v)$ is at most the length of the shortest walk from $s$ to $v$ consisting of at most $i$ edges. So, by the end of the first for loop, every vertex reachable from $s$ will have finite length. At least one edge from every negative cycle reachable from $s$ would be tense in every subsequent iteration with normal relaxations, because the $\text{dist}(u) + w(u \to v)$ sums around the cycle have to come to something smaller than the distance to the first vertex along the cycle. Therefore, we will discover a vertex
for which to assign distance $-\infty$ in the first iteration of the second for loop. Let $w$ be this vertex.

Any vertex $v$ reachable from $w$’s cycle is also reachable from $w$ and therefore, there is a walk from $s$ to $w$ to $v$, and we should assign $\text{dist}(v) = -\infty$. We may now prove inductively that after the $i+1$st iteration of the 2nd for loop, every vertex $v$ for which there is a path of at most $i$ edges from $w$ to $v$ will have $\text{dist}(v) = -\infty$. Indeed, $\text{dist}(w) = -\infty$ after 1 iteration as explained above. Let $v$ be a vertex for which there is a path of $i$ edges $w \rightarrow v_1 \rightarrow \ldots \rightarrow u \rightarrow v$ from $w$ to $v$. Either $\text{dist}(v) = -\infty$ before the $i+1$st iteration, or $\text{dist}(u) = -\infty$ inductively and the $i+1$st iteration will set $\text{dist}(v)$ to be $-\infty$ when it loops over edge $u \rightarrow v$. ■

**Rubric:** 7 points total: 4 points for the algorithm; 2 point for the proof; 1 point for the analysis.