Let $G = (V, E)$ be a directed graph with weighted edges; edge weights could be positive, negative or zero.

(a) (4 points) Describe an $O(n^2)$ algorithm that takes as input $v \in V$ and returns an edge-weighted graph $G' = (V', E')$ such that $V' = V \setminus \{v\}$ and the shortest path distance from $u$ to $w$ in $G'$ for any $u, w \in V'$ is equal to the shortest path distance from $u$ to $w$ in $G$.

Solution: INSERT YOUR SOLUTION HERE

(b) (4 points) Now assume that you have already computed the shortest distance for all pairs of vertices in $G'$. Give an $O(n^2)$ algorithm that computes the shortest distance in $G$ from $v$ to all nodes in $V'$ and from all nodes in $V'$ to $v$.

Solution: INSERT YOUR SOLUTION HERE

(c) (4 points) Use part (a) and (b) to give a recursive $O(n^3)$ algorithm to compute the shortest distance between all pairs of vertices $u, v \in V$.

Solution: INSERT YOUR SOLUTION HERE
A looped tree $G = (V, E)$ is an edge-weighted directed graph built from some (directed) binary tree $T$ on $V$ rooted at some node $r \in V$, by adding an edge from every leaf in $T$ back to $r$ (e.g., if $T$ was a long directed path, the looped tree $G$ would be a cycle). Assume that the vertices are labeled from $1, \ldots, n$, with the root $r$ having label $1$, and the edges are given in the form of an adjacency list, along with the corresponding edge weights. Moreover, assume that all the edge weights are non-negative.

Your goal in this problem be to develop a faster-than-Dijkstra single-source shortest-path algorithm computing all shortest distances $d[v]$ from a given input node $u \in V$ to all other nodes $v$ of $G$. You will do it by solving the following sub-problems:

(a) (1 point) Show that the number of edges in $G$ is $O(n)$.

**Solution:** INSERT YOUR SOLUTION HERE

(b) (2 points) Compute the running time of Dijkstra’s algorithm to compute the shortest distance $d[v]$ from a given vertex $u$ to all $v \in V$.

**Solution:** INSERT YOUR SOLUTION HERE

(c) (3 points) Modify the BFS algorithm from the lecture appropriately to give an $O(n)$ algorithm that computes for all $v \in V$ the shortest distance $c[v]$ from the root $r$ to $v$.

**Solution:** INSERT YOUR SOLUTION HERE

(d) (5 points) Give an $O(n)$ algorithm that computes the shortest distance $\alpha$ from $u$ to the root $r$, for the given source vertex $u$. (Hint: Think of recursion, but make sure you terminate, and fast!)

**Solution:** INSERT YOUR SOLUTION HERE

(e) (3 points) Let $T = (V, E')$ be the original rooted tree you started from (before adding the edges from the leaves of $T$ to $r$). Give an $O(n)$ algorithm to compute the shortest distance $b[v]$ from $u$ to $v$ in $T$ (not $G$), for all $v \in V$.

**Solution:** INSERT YOUR SOLUTION HERE

(f) (4 points) In $G$, express (with proof) the shortest distance $d[v]$ from $u$ to $v$ in terms of $b[v], c[v]$, and $\alpha$. Use this expression to obtain an algorithm to compute $d[v]$ for all $v \in V$ with running time asymptotically faster than Dijkstra’s algorithm.

**Solution:** INSERT YOUR SOLUTION HERE
Assume we define the length of a path to be the maximum weight among all the edges of the path rather than the sum of all the edge weights. Argue that the Floyd-Warshall algorithm can be modified to handle this situation. That is, for every pair of nodes, find the value of the “shortest” path connecting these nodes, where the meaning of “shortest” is appropriately modified according to the above. Make sure to write the pseudo-code of your new algorithm and briefly argue why it is correct.

Solution: INSERT YOUR SOLUTION HERE