A **looped tree** is a weighted, directed graph built from a binary tree by adding an edge from every leaf back to the root. Every edge has non-negative weight.

(a) How much time would Dijkstra’s algorithm require to compute the shortest path between two vertices $u$ and $v$ in a looped tree with $n$ vertices?

**Solution:** A looped tree with $n$ vertices has at most $n$ leaves. Every vertex except the root has in-degree 1 which has in-degree equal to the number of vertices. Therefore, there are $O(n)$ edges, and Dijkstra’s algorithm would run in $O(n \log n)$ time.

**Rubric:** 3 points total.

(b) Describe and analyze a faster algorithm. You may describe it using English prose, pseudocode, or some combination of the two.

**Solution:** We’ll sketch an algorithm to compute the shortest path from $u$ to every other vertex. First, we compute the shortest paths from $u$ to its tree descendants by running the shortest paths algorithm for a DAG in the subtree rooted at $u$. If $u$ is the root, we are done, because no shortest path will go through $u$ a second time.

Otherwise, we then compute the distance to the root by finding the minimum sum of distances to a leaf descendant of $u$ plus length of edge from that leaf to the root. We set the root’s predecessor as the leaf that gave us that minimum. Finally, we run the DAG algorithm again from the root staring with the distance computed instead of 0 and in the subgraph that is missing edges going into the root and the entire subtree rooted at $u$.

Each of the three steps, computing shortest paths to descendants of $u$, computing the best edge into the root, and computing shortest paths directly from the root, take $O(n)$ time. Therefore, the whole algorithm takes $O(n)$ time.

**Rubric:** 7 points total: 5 points for algorithm description; 2 points for running time analysis.
(a) Describe and analyze an algorithm to compute the maximum weight spanning tree of a given edge-weighted undirected graph $G = (V,E)$.

**Solution:** The algorithms we covered in class work just fine with negative weight edges, so we’ll negate the weight of every edge and then run, say, Kruskal’s algorithm in the resulting graph. The whole process takes $O(|E| \log |V|)$ time.

**Rubric:** 4 points total: 3 points for the algorithm description; 1 point for the running time analysis. Part is still worth 4 points if student uses a faster algorithm for minimum spanning trees.

(b) Describe and analyze a fast algorithm to compute a minimum-weight feedback edge set of a given edge-weighted undirected graph $G = (V,E)$. You may assume $G$ is connected and that edges have non-negative weight.

**Solution:** Every edge has non-negative weight, so we want to remove as few edges as possible. In particular, the result of removing the minimum weight feedback edge set $F$ should be connected, as we could otherwise reduce its weight by removing at least one edge from $F$ that connects two components of $G - F$. Graph $G - F$ is acyclic as well, so it must be a spanning tree. It should be the heaviest spanning tree possible so that $F$ is as light as possible.

We’ll compute a maximum weight spanning tree $T$ and then return $E \setminus T$. Computing the tree and taking the compliment of its edges takes $O(|E| \log |V|)$ time total.

**Rubric:** 6 points total. 4 points for the algorithm description; 2 point for the running time analysis.