Due: Wednesday 9/19 before 9:15

1. **(4.10)** Let $G = (V, E)$ be an undirected graph with a nonnegative edge cost function c. Assume you are given a minimum-cost spanning tree T in G. Now assume that a new edge is added to G, connecting two nodes v and w with cost c.

   (a) Give an efficient algorithm to test if T remains the minimum-cost spanning tree with the new edge added to G. Make your algorithm run in time $O(|E|)$. Can you do it in $O(|V|)$ time? Please note any assumptions you make about what dada structure is used to represent the tree T and the graph G.

   (b) Suppose T is no longer the minimum-cost spanning tree. Give a linear-time algorithm (time $O(|E|)$) to update the tree T to the new minimum-cost spanning tree.

2. **(4.17)** Consider the following variation on the Interval Scheduling Problem. You have a processor that can operate 24 hours a day, every day. People submit requests to run *daily jobs* on the processor. Each such job comes with a *start time* and an *end time*. If the job is accepted to run on the processor, it must run continuously, every day, for the perod between its start and end times. (Note that certain jobs can begin before midnight and end after midnight; this makes for a type of situation different from what we saw in the Interval Scheduling Problem).

   Given a list of n such jobs, your goal is to accept as many jobs as possible (regardless of their length), subject to the constraint that the processor can run at most one job at any given point in time. Provide an algorithm to do this with a running time that is polynomial in n. You may assume for simplicity that no two jobs have the same start or end times.

   Example. Consider the following four jobs, specified by `(start-time, end-time)` pairs:

   (6pm, 6am), (9pm, 4am), (3am, 2pm), (1pm, 7pm).

   The optimal solution would be to pick the two jobs (9pm, 4am) and (1pm, 7pm), which can be scheduled without overlapping.

3. **(4.18)** Your friends are planning an expedition to a small town deep in the Canadian north next winter break. They've researched all the travel options and have drawn up a directed graph whose nodes represent intermediat destinations and edges represent the reoads be-tweeen them.

   In the course of this, they've also learned that extreme weather causes roads in this part of the world to become quite slow in the winter and may cause large travel delays. They've found an excellent travel Web site that can accurately predict how fast they'll be able to travel along the roads; however, the speed of travel depends on the time of the year. More precisely, the Web site answers queries of the following form: given an edge $e = (u, v)$ connecting two sites u and v, and given a proposed starting time t from location u, the site will return a value $f_e(t)$, the predicted arrival time at v. The web site guarantees that

$f_e(t) \geqslant t$ for every edge e and every time t (you can't travel backward in time), and that $f_e(t)$ is a monotone increasing function of t (that is, you do not arrive earlier by starting later). Other than that, the functions f_e may be arbitrary. For example, in areas where the travel time does not vary with the season, we would have $f_e(t) = t + \ell_e$, where ℓ_e is the time needed to travel from the beginning to the end of the edge e.

Your friends want to use the Web site to determine the fastest way to travel through the directed graph from their starting point to their intended destination. (You should assume that they start at time 0 and that all predictions made by the Web site are completely correct.) Give a polynomial-time algorithm to do this, where we treat a single query to the Web site (based on a specific edge e and a time t) as taking a single computational step.

4. **(4.27)** In trying to understand the combinatorial structure of spanning trees, we can consider the space of *all* possible spanning trees of a given graph and study the properties of this space. This is a strategy that has been applied to many similar problems as well.

   Here is one way to do this. Let G be a connected graph, and T and T' two different spanning trees of G. We say that T and T' are *neighbors* if T contains exactly one edge that is not in T', and T' contains exactly one edge that is not in T.

   Now, from any graph G, we can build a (large) graph $\mathcal{H}$ as follows. The nodes of $\mathcal{H}$ are the spanning trees of G, and there is an edge between two nodes of $\mathcal{H}$ if the corresponding spanning trees are neighbors.

   Is it true that, for any connected graph G, the resulting graph $\mathcal{H}$ is connected? Give a proof that $\mathcal{H}$ is always connected, or provide an example (with explanation) of a connected graph G for which $\mathcal{H}$ is not connected.