Professor Borden proposes a new divide-and-conquer algorithm for computing minimum spanning trees, which goes as follows. Given a graph \( G = (V, E) \), partition the set \( V \) of vertices into two sets \( V_1 \) and \( V_2 \) such that \( |V_1| \) and \( |V_2| \) differ by at most 1. Let \( E_1 \) be the set of edges that are incident only on vertices in \( V_1 \) and let \( E_2 \) be the set of edges that are incident only on vertices in \( V_2 \). Recursively solve a minimum-spanning-tree problem on each of the two subgraphs \( G_1 = (V_1, E_1) \) and \( G_2 = (V_2, E_2) \). Finally select the minimum-weight edge in \( E \) that crosses the cut \( (V_1, V_2) \) and use this edge to unite the resulting two minimum spanning trees into a single spanning tree.

Either argue that the algorithm correctly computes a minimum spanning tree of \( G \) or provide an example for which the algorithm fails.

Solution: ****************** INSERT YOUR SOLUTION HERE ******************
Jack loves hiking and wants to always try different hiking trails. He has a map with \( n \) different landmarks labeled 1, \ldots, \( n \). Since there will be no water available during the paths between landmarks, the map also carries information about the amount of water \( w[i,j] \) you need to carry to survive while taking the direct road from landmark \( i \) to \( j \) (without any intermediate landmarks in between), for \( 1 \leq i, j \leq n \).

On a given day, Jack may start at any landmark \( i \) and end at any landmark \( j \) for \( 1 \leq i, j \leq n \). He does not mind to visit some intermediate landmarks on the way, as long as the maximum weight that he needs to carry during the entire journey to be as little as possible. You may assume that there is plenty of water to fill from at each of the landmarks.

Assume you are given an edge-weighted graph \( G = (V, E) \), where the set of vertices correspond to the \( n \) landmarks and the weight of the edge from \( i \) to \( j \) denotes the amount of water needed when taking the direct road from \( i \) to \( j \).

(a) (6 points) Assume that \( G \) is an undirected tree, i.e., the graph is an undirected connected graph with no cycles. Design an \( O(n^2) \) algorithm to help Jack determine the smallest possible maximum weight \( s[i,j] \) that he needs to carry when going from \( i \) to \( j \), for all pairs of vertices \( i, j \).

Solution: ***************** INSERT YOUR SOLUTION HERE ************ *

(b) (7 points) Now give an \( O(n^2) \) algorithm to solve the problem of part (a) for any connected undirected graph \( G \), and not necessarily a tree. Make sure you prove the correctness of your algorithm, and not just the run-time. (Hint: Think of MST.)

Solution: ***************** INSERT YOUR SOLUTION HERE ************ *

(c) (5 points) Now assume that you are given a directed graph \( G = (V, E) \), so that \( w[i,j] \) might be different from \( w[j,i] \) (think that going uphill might require much more water than going downhill). Give an \( O(n^3) \) algorithm that solves the problem in part (a). (Hint: Modify Floyd-Warshall.)

Solution: ***************** INSERT YOUR SOLUTION HERE ************ *
Solutions to Problem 3 of Homework 12 (20 points)

You are given a directed graph \( G = (V, E) \) with non-negative edge weights \( w(u, v) \) representing the time it takes to go from a node \( u \) to \( v \) when \( u \) and \( v \) are directly connected. (We assume all \( w(u, v) < \infty \) for simplicity.) Some subset \( F \) of vertices in \( G \) are pharmacies, while some other subset \( D \) are doctor offices. You live at a node \( s \) and have a medical emergency. You now need to go to some doctor office \( d \in D \) to get a prescription for the drug, then to some pharmacy \( f \in F \), and finally back home. Recall that a proper output must be the full path described above. However some paths are better then others. Solve the following variants of this problem, where each variant describes how to compare different valid paths when choosing the optimal path you are looking for. Each variant could be solved by one “clever” run of the Dijkstra’s algorithm, on an appropriate modification of our graph \( G \). You have to explain your choice and state the resulting running time as a function of \( n \) (remember, we assume \( m = \Theta(n^2) \) here, as \( w(u, v) < \infty \)).

(a) (3 points) Assume only the distance from your home \( s \) to the doctor’s office matters (e.g., you need to get doctor’s emergency help asap, and the time it then takes to the pharmacy and back home are not important).

**Solution:** ***************** INSERT YOUR SOLUTION HERE *****************  

(b) (5 points) Assume only the distance between the doctor office \( d \) and the pharmacy \( f \) matters. E.g., the doctor would perform some painful procedure, but does not have a tranquilizer to numb the subsequent pain. Thus, you must choose an office \( d \in D \) and pharmacy \( f \in F \) with the smallest distance between them, but you do not care how far \( d \) is from \( s \) or \( s \) is from \( f \). (E.g., if there is a pharmacy in some doctor’s office in Antarctica, you do not mind going there, as the answer you care will be 0, even if you live in NYC.)

(\textbf{Hint}: Add an artificial source node \( s' \) to \( G \) and compute the distance from \( s' \) to “fill the blank” in your new graph.)

**Solution:** ***************** INSERT YOUR SOLUTION HERE *****************  

(c) (5 points) Assume only the distance from the pharmacy \( f \) back to home \( s \) matters. E.g., the medicine must be administered by the pharmacist and has some quick side effect, so you must get to bed asap after taking the medicine in the pharmacy.

(\textbf{Hint}: Two solutions are possible: one does something to the edges of \( G \), and the other again adds some artificial source similar to part (b).)

**Solution:** ***************** INSERT YOUR SOLUTION HERE *****************  

(d) (7 points) Finally, assume that the overall trip time from \( s \) to \( d \) to \( f \) to \( s \) matters. Show how to combine the ideas in parts (a)-(c) to solve this variant.

**** INSERT YOUR NAME HERE ****, Homework 12, Problem 3, Page 1
(Hint: Make several “copies” of $G$ and connect them “appropriately” by 0-weight edges. The copies will ensure that every valid path must pass through a doctor office followed by a pharmacy.)

Solution: **************** INSERT YOUR SOLUTION HERE ***************
A looped tree $G = (V, E)$ is an edge weighted directed graph built from a 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. Also, assume that all the edge weights are non-negative.

Your goal will 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 $v$ for all $v \in V$.

Solution: **************** INSERT YOUR SOLUTION HERE ************

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

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) 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 the Dijkstra’s algorithm.

Solution: **************** INSERT YOUR SOLUTION HERE ************
You are given a directed graph $G = (V, E)$ representing some financial choices. Each edge $(u, v) \in E$ has a weight $w(u, v)$, where $w(u, v) > 0$ represents a cost, and $w(u, v) < 0$ represents a profit. Your initial portfolio is a vertex $s \in V$, and at each step you are allowed to go from your current node $u \in V$ to a neighboring node $v \in \text{Adj}(u)$, incurring a cost $w(u, v)$ if $w(u, v) > 0$, or a profit $-w(u, v)$ otherwise.

(a) (4 points) We say that a vertex $s$ is super-lucky if $s$ itself is part of a cycle $C$ of negative weight, so that starting from $s$ one can repeatedly come back to $s$ with some profit. Using the “matrix multiplication” approach, design $O(n^3 \log n)$ algorithm to find all super-lucky vertices.

Solution: ****************** INSERT YOUR SOLUTION HERE ******************

(b) (4 points) Say that $s$ is lucky if there exists a way to eventually make unbounded profit starting from $s$ (but not necessarily coming back to $s$ infinitely many times as with super-lucky vertices). Give the fastest algorithm you can for finding all lucky vertices (from scratch, without assuming you already solved part (a)).

Solution: ****************** INSERT YOUR SOLUTION HERE ******************

(c) (4 points) Solve the same problem as in part (b), but assuming somebody already gave you for free the list of all super-lucky vertices (or, alternatively, you already ran your solution in part (a), and want to use it to compute lucky vertices faster). Namely, give the fastest algorithm you can think of for finding all lucky vertices given all super-lucky vertices. (Hint: Make sure you use super-lucky vertices instead of computing from scratch!)

Solution: ****************** INSERT YOUR SOLUTION HERE ******************

(d) (4 points) Assume $s$ is not lucky (and, hence, not super-lucky). Design a fast algorithm to compute the best finite “financial strategy” to make as much profit starting from $s$ as possible. State the running time of your algorithm. (Hint: Think Bellman-Ford.)

Solution: ****************** INSERT YOUR SOLUTION HERE ******************