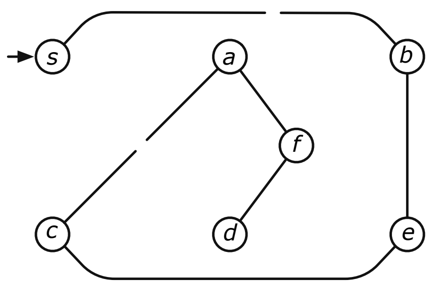
CS 6363.003 Spring 2021	Midterm 2 Problem 1 Solution

Clearly indicate the edges of the following spanning trees of the weighted graph by, say, drawing a new copy of the graph with vertex labels and edge weights intact.
Each part is worth 2.5 points out of 10.

(a) A depth-first spanning tree rooted at *s*

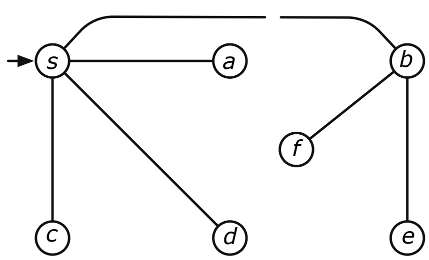
Solution:



■

(b) A breadth-first spanning tree rooted at *s*

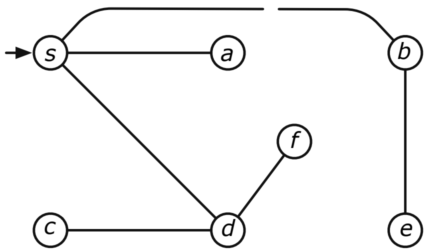
Solution:



■

(c) A shortest paths tree rooted at s

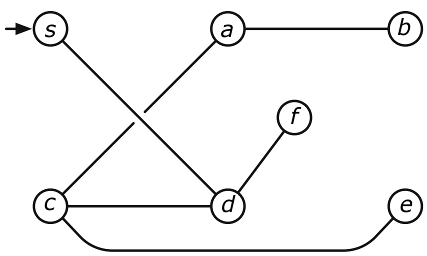
Solution:



■

(d) A minimum spanning tree

Solution:



■

CS 6363.003 Spring 2021	Midterm 2 Problem 2 Solution

Suppose you are a shopkeeper living in a country with n different types of coins, with values $1 = c[1] < c[2] < \cdots < c[n]$. Not wanting to burden their pockets, whenever you give a customer change, you always use the smallest possible number of coins.

(a) **(3 out of 10)** Show there is a set of coin values for which the greedy algorithm does *not* always give the smallest possible number of coins. Please state both the coin values and the amount of change C for which the greedy algorithm is not optimal.

Solution: Consider the coin values $\{1, 5, 6\}$. If asked to make $C = 10$ units of the changes, the greedy algorithm would use five coins: $\langle 6, 1, 1, 1, 1\rangle$. However, it is possible to use only two coins: $\langle 5, 5\rangle$. ■

(b) **(7 out of 10)** Suppose your country's government decides to impose a currency system where the coin denominations are consecutive powers $b^0, b^1, b^2, \ldots, b^k$ for some integer $b \geq 2$. Prove the greedy algorithm described in part (a) does make optimal change in this currency system.

Solution: First, observe that for any $0 \leq i \leq k-1$, the value b^{i+1} is an integer multiple of b^i. Inductively, one cannot exceed the value b^i using smaller coins without meeting it, so one cannot exceed b^{i+1} using smaller coins without meeting it as well.

Now, suppose we're asked to make change C, and let i be the largest integer between 0 and k such that $b^i \leq C$. Suppose the optimal method does not use coin b^i. From the above, we see one can add up the coins in the optimal set in any order, and at a certain moment meet b^i exactly. We can replace all the coins up to this moment with the single coin b^i, contradicting the optimality of the coin set that excludes b^i.

Finally, we see the optimal change for C includes b^i. Inductively, the greedy algorithm goes on to make the remaining change $C - b^i$ optimally. ■

(a) **(3 out of 10)** Let $G = (V, E)$ be a connected undirected graph with real edge weights $w : E \to \mathbb{R}$. Briefly describe and analyze a fast algorithm to compute a *maximum* weight spanning tree of G. You may assume edge weights are distinct. You do not need to justify correctness of your algorithm.

Solution: We define the edge weight function $w' : E \to \mathbb{R}$ where $w'(e) := -w(e)$. Then, we compute and return a *minimum* spanning tree under weight function w' using any algorithm from class such as Kruskal's. Computing the minimum spanning tree is the bottleneck, so the algorithm runs in $O(E \log V)$ time.

■

Explanation Any spanning tree T of weight $w(T)$ under w weights $-w(T)$ under w'. Therefore, minimizing the weight of T under w' is equivalent to maximizing it under w.

(b) **(7 out of 10)** Let $G = (V, E)$ be a connected undirected graph with *positive* edge weights $w : E \to \mathbb{R}_{>0}$. A ***feedback edge set*** of G is a subset F of its edges such that every cycle in G contains at least one edge in F. In other words, removing every edge in F turns the graph G into a forest. Briefly describe and analyze a fast algorithm to compute a minimum weight feedback edge set of G. You may assume edge weights are distinct, and you may assume your solution to part (a) is correct. You do not need to justify correctness of your algorithm.

Solution: We compute a maximum weight spanning tree T using the algorithm from part (a) and then return $E - T$. The algorithm from part (a) is the bottleneck, so this procedure takes $O(E \log V)$ time as well. ■

Explanation Let F be a minimum weight feedback edge set. The graph $G-F$ is connected; otherwise, we could take any edge e connecting two components of $E \setminus F$ and remove it from F, reducing the weight of F while still leaving $G - F$ as a forest. Therefore, $G - F$ is a spanning tree. Minimizing the weight of feedback edge set F is equivalent to maximizing the weight of spanning tree $G - F$.

CS 6363.003 Spring 2021	Midterm 2 Problem 4 Solution

Suppose we are given a directed acyclic graph $G = (V, E)$ with real edge lengths $\ell : E \to \mathbb{R}$ such that G has a unique source s and a unique sink t. In other words, s is the only vertex with no predecessor and t is the only vertex with no successor. We have also been given a non-negative integer k and told that some vertices of G have been marked *important*. For simplicity, we may assume neither s nor t have been marked important.

Our goal is to design an algorithm to find the maximum length over all paths from s to t that contain at least k important vertices.

(a) **(5 out of 10)** For any vertex $v \in V$ and any non-negative integer r, let $MaxLength(v, r)$ denote the maximum length over all paths from v to t that contain at least r important vertices. If no such path exists, let $MaxLength(v, r) := -\infty$.

Give a recursive definition of $MaxLength(v, r)$. You do not need to justify correctness of your definition.

Solution: For simplicity, we define a max over no elements to be $-\infty$.

$$MaxLength(v,r) = \begin{cases} 0 & \text{if } v = t \text{ and } r = 0 \\ -\infty & \text{if } v = t \text{ and } r > 0 \\ \max_{v \to w}(\ell(v \to w) + MaxLength(w, 0)) & \text{if } v \neq t \text{ and } r = 0 \\ \max_{v \to w}(\ell(v \to w) + MaxLength(w, r)) & \text{if } v \neq t,\ r > 0 \text{ and } v \text{ is not important} \\ \max_{v \to w}(\ell(v \to w) + MaxLength(w, r-1)) & \text{otherwise} \end{cases}$$

■

(b) **(5 out of 10)** Describe and analyze an efficient dynamic programming algorithm to compute the maximum length over all paths from s to t that contain at least k important vertices. If no such path exists, your algorithm should return $-\infty$.

Solution: We want to solve $MaxLength(v, r)$ for all $v \in V$ and values $0 \leq r \leq k$. Each subproblem $MaxLenght(v, r)$ depends upon those taking a vertex w such that $v \to w$ is an edge. Therefore, we can iterate over all vertices in postorder and all r in arbitrary order. We return the value $MaxLength(s, k)$.

For each of the $O(k)$ choices of r, we solve one subproblem per vertex and have one dependency per edge. Therefore, the algorithm will run in $O(k(V + E))$ time. ■