*Problem* 1 (10 Points). Sometimes, there are many shortest paths between nodes (i.e. multiple paths that are at least as short as any other path). Show how to compute the number of shortest paths from a specific node $v$ to each other node. Your algorithm should have the same running time as Dijkstra's algorithm

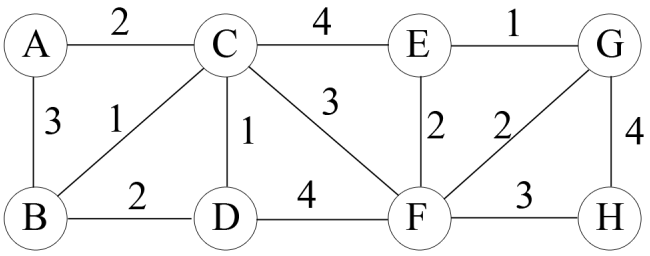
*Problem* 2 (10 Points). Suppose that, in addition to weights on edges $weight(u, v)$, we also have weights on nodes $weight(u)$, and the length of a path is the sum of the weights of the edges and nodes on the path (including endpoints). We are now interested in computing the shortest path between two nodes using this new notion of length.

(a) In a directed graph, show how to reduce this problem to the standard shortest path problem. That is, show how to give new weights to edges $weight'(u, v)$ such that a shortest path in the standard shortest path problem using weights $weight'(u, v)$ is the same as the shortest path in the more general problem using weights $weight(u, v)$ and $weight(u)$. How are the lengths of the path under the two measures related?

(a) Do the same for an undirected graph.

*Problem* 3 (0 Points). Removed

*Problem* 4 (10 Points). Consider the following graph:




(a) In what order does Prim's algorithm add edges to the MST? Whenever there is a choice of nodes, choose the one that comes alphabetically first. Whenever there is a choice of edges, choose the one with the alphabetically first endpoint (if both share an alphabetically first endpoint, use the alphabetical ordering on the other end point).

(b) In what order does Krustkal's algorithm add edges to the MST? Use the same method for breaking ties as above.

*Problem* 5 (30 Points). The following statements may or may not be correct. In each case, either prove it (if it is correct) or give a counter example (it it isn't correct). Always assume the graph $G = (V, E)$ is undirected and connected. Do not assume that edge weights are distinct unless this is specifically stated.

(a) If graph $G$ has more than $|V| - 1$ edges, and there is a unique heaviest edge, then this edge cannot be part of a minimum spanning tree.

(b) If $G$ has a cycle with a unique heaviest edge $e$, then $e$ cannot be part of any MST.

(c) Let $e$ be any edge of minimum weight in $G$. Then $e$ must be part of some MST.

(d) If the lightest edge in a graph is unique, then it must be part of every MST.

(e) If $e$ is part of some MST of $G$, then it must be a lightest edge across some cut of $G$.

(f) If $G$ has a cycle with a unique lightest edge $e$, then $e$ must be part of every MST

(g) The shortest-path tree computed by Dijkstra's algorithm is necessarily an MST.

(h) The shortest path between two nodes is necessarily part of some MST.

(i) Prim's algorithm works correctly when there are negative edge weights

(j) (For any $r \geq 0$, define an $r$-path to be a path whose edges all have weight $< r$). If $G$ contains an $r$-path from node $s$ to $t$, then every MST of $G$ must also contain an $r$-path from node $s$ to $t$.

*Problem* 6 (10 Points). In a spanning tree, a bottleneck is an edge with highest weight. A spanning tree is a *minimum bottleneck spanning tree*, or MBST, if no other spanning tree of the graph has a smaller bottleneck.

(a) Show that every minimum spanning tree is also a minimum bottleneck spanning tree.

(b) Is every minimum bottleneck spanning tree a minimum spanning tree? Prove or give a counter example.

*Problem* 7 (10 Points). A small business — say, a photocopying service with a single large machine — faces the following scheduling problem. Each morning, they get a set of jobs from customers. They want to do the jobs on their single machine in an order that keeps their customers happiest. Customer $i$'s job will take $t_i$ time to complete. Given a schedule (i.e. an ordering of the jobs), let $C_i$ denote the finishing time of job $i$. For example, if job $j$ is the first to be done, we would have $C_j = t_j$, and if job $j$ is done right after job $i$, we could have $C_j = C_i + t_j$. Each customer also has a given weight $w_i$ that represents his or her importance to the business. The happiness of customer $i$

is expected to be dependent on the finishing time of $i$'s job. So the company decides that they want to order the jobs to minimize the weighted sum of the completion times, $\sum_{i=1}^{n} w_i C_i$.

Design an efficient algorithm to solve this problem. That is, you are given a set of $n$ jobs with a processing time $t_i$ and a weight $w_i$ for each job. You want to order the jobs so as to minimize the weighted sum of completion times, $\sum_{i=1}^{n} w_i C_i$.

Total points: 80