James Sweetman (jts2939)
Jung Yoon (jey283)
Unique #51835

HW #3

Q1

a. (Smallest Time) - Counterexample

Let's say that jobs with the longest duration times have the lowest weights for a particular example. That is, when ordered in the order of smallest time ascendingly, it's doubles as a list of jobs ordered according to weight ascendingly. For such a case, this would mean that the jobs with the largest weights (more important) would also have the largest completion times since they would be scheduled in the order of weights as well, thus making the the summation of C*w unminimized. To be more specific, let's consider the following example below. Note that even simply switching Job 4 and Job 5 in the schedule would make it more optimal so we know that this is a valid counterexample.

Order	Job 1	Job 2	Job 3	Job 4	Job 5
Duration	1	2	3	4	5
Weight	1	10	20	50	100

b. (Most Important First) - Counterexample

An example to prove that the most important first will not provide the optimal solution is given below. Note that it would be more optimal if Job 3, Job 4 were scheduled before Job 1, 2 and 5 so we know that using straight most important first wouldn't give minimize the summation of C*w.

Order	Job 1	Job 2	Job 3	Job 4	Job 5
Duration	80	30	3	2	1
Weight	100	50	20	10	1

c. Maximum time-scaled importance first: Proof

Unlike the previous two methods that only took into account either duration or weight, Maximum time-scaled importance first ordering takes into account both the weight and duration of each job, which is what we need to get the optimal schedule. We know that to get the optimal solution that we must have it so that the summation of C*w is minimized. We have pre-set weights w and can not control them but we can control the completion times C by choosing an optimal scheduling algorithm. That is, we need to choose an algorithm that will minimize C for each w such that the jobs are scheduled in an order in

which we get the most optimal combination of weight and completion time. By using a time-scaled importance first algorithm, we are basically scheduling in the order of most weighted and least duration time dually.

Q2 (a)

The algorithm doesn't work with weighted nodes because it will terminate once the destination node is reached, meaning it will not re-evaluate and check other paths to the destination node. The reason the algorithm does this is because it assumes that the weight to each node is the lowest possible cost due to it's greedy orientation. However, this assumption will only work with positive weights as that asserts the minimal cost of the path will not be changed down the road. By using negative weights, you could have a negative cycle that will constantly reduce the weight cost and cause the algorithm to never terminate. Also, the algorithm would produce the wrong result if the algorithm reached the end result without checking every other path. For example, let G be an undirected graph with nodes A, B, and C with edges A-B (+3), B-C(-13), and A-C (+2). If our source node in G was A and we wanted to find the minimal distance to destination C, the algorithm would produce A-B as the answer even though going from A to B to C would produce a cost of -10.

Q2 (b)

This algorithm will be an adaptation of Dijkstra's greedy shortest path algorithm. Nodes and edges will refer to cities and flights, respectively.
Assume that the times are based on a 24-hour clock (i.e. 11 pm is 23:00) and only the hour time is being used.
Given graph G(V, E)
Let S be the set of explored nodes

For each $u \in S$, we store two times d(t) and a(t)

Initially $S = \{s\}$ and $d(s) = 0$ where s is the starting city
While $S \neq V$

Select a destination node $v \notin S$ with at least one edge from some source node $q \in S$ with the earliest tentative destination time if and only if the source city's flight leaves at least an hour after arrival time to the source otherwise known as a(q).
Add v to S and define a(v) = a'(v).

EndWhile

We prove this by induction on the size of S very similarly to how the proof is done for Dijkstra's Algorithm.. The case $|S| = 1$ is easy, since then we have $S = \{s\}$ and $a(s) = 0$. Suppose the claim holds when $|S| = k$ for some value $k \geq 1$. We now grow S to size k+1 by adding the node v. Let (u,v) be the final edge on our s-v path P_v

By induction hypothesis, P_u is the shortest path from the starting node to u for every u in S. We now must show any other path to v is as long as P_v. Let y be the first node on some other path P that is not in S and x be some node in S just before y. P cannot be shorter than P_v because it is already at least as long as P_v by the time it leaves S. The algorithm must have considered adding node y to the set S and rejected it in favor of v. This means that there is no path from s to y through x that is shorter than P_v.

Q3 (a)

Our algorithm will be a version of Kruskal's Algorithm but instead of finding a Minimum Spanning Tree for the sum, we're looking for the Minimum-product spanning tree. From Kruskal's proof, we know that an optimal MST is produced. Now we must prove that it will be the minimum product. Let a, b, c, and d be edges within the graph. Assume a + b < b + c. Then we know a < b and if we multiply by c instead then ac < bc. Through this we know that the same edges picked in the normal MST for a sum will also work for the product.

Q3 (b)

Let's say that there are two MSTs for G called T1 and T2 for our purposes. Then, let edge e1 be the minimum weighted edge in G such that e is the edge connected node a to b, that such that it is the minimum weighted edge in either T1 but not T2.

We start by adding e to T2 but that would create a cycle K in T2. This means that there is at least one edge e2 in K that is not in T1 because T1 would also have a cycle otherwise. Thus, we know that node a in e1 <= node a in e2 and since we know that the edge weights are distinct, we also know that a in e1 < a in e2.

Now, replacing e2 with e1 within T2, we get an entirely new spanning tree with total weight that is less than T2 was thought to be, thus giving us a contradiction. Thus, if the weights on the edges of a weighted undirected graph G are such that no two edges have the same weight (i.e. all edgeweights are unique) and all weights are positive, then G has a unique minimum spanning tree.

Q3 (c)

First we must note that a minimum spanning tree is a spanning tree with weight less than or equal to the weight of every other spanning tree. We must also note the cycle property which states that for any cycle C in the graph, if the weight of an edge e of C is larger than the weights of all other edges of C, then this edge cannot belong to an MST. Then, since we know that all of T's edges e are at least in one MST in the set of

all MSTs for G, we can be sure by the cycle property that T contains no edges e that cannot belong to an MST by the cycle property. This is to say that T contains only edges e that could be in MSTs so T is an MST it in of itself and not just a spanning tree.