Exam 4

CS 3510A, Spring 2014

Instructions:

- **Do not open this exam packet until you are instructed to.**
- **Write your name** on the line below.
- You will have **50 minutes** (from 11:05 to 11:55) to earn **50 points**. Pace yourself!
- You may use one double-sided 8.5-by-11" study sheet. **You may not use any other resources, including electronic devices.**
- Do your best to fit your final answers into the space provided below each question. You may use the backs of the exam pages, and the final page of the exam, as scrap paper.
- **You may refer to any facts from lectures or the textbook without re-deriving them.**
- Your work will be graded on correctness and clarity. Please write legibly!

Question	Points	Score
1	15	
2	10	
3	10	
4	15	
Total:	50	

Your name: ______________________________

1. (15 points) *True or False.* State whether each of the following sentences is true or false, **and give a convincing justification for your answer**. A correct justification is worth more than a correct true/false answer.

   (a) The following is a correct algorithm to topologically sort a DAG: repeatedly run BFS from an arbitrary unvisited vertex until all vertices have been visited (i.e., colored black), then output the vertices in the order they were visited.

   Solution: False. We can show this with a simple counterexample: consider the graph consisting of a single edge $u \to v$. If we run BFS starting from v, then v is visited first and is immediately finished. Then u is visited (and v is *not* re-visited, because it was colored black when it was first visited). The above algorithm would therefore order the graph as $v \leftarrow u$, but this is not a topological ordering because it has a left-pointing edge.

   (b) If $u \overset{p}{\rightsquigarrow} v \overset{q}{\rightsquigarrow} w$ is a shortest path from u to w, then $v \overset{q}{\rightsquigarrow} w$ must be a shortest path from v to w. (The notation $a \overset{r}{\rightsquigarrow} b$ indicates a particular path r from a to b.)

   Solution: True. (This is the "shortest subpaths" property from the book.) For if $v \overset{q}{\rightsquigarrow} w$ was *not* a shortest path from v to w, then there would be a strictly shorter path $v \overset{q'}{\rightsquigarrow} w$. But then $u \overset{p}{\rightsquigarrow} v \overset{q'}{\rightsquigarrow} w$ would be a strictly shorter path from u to w than the given one, a contradiction.

   (c) If a directed graph contains a path from vertex u to vertex v, then *any* execution of DFS must discover v before finishing u, i.e., $v.d \leq u.f$.

   Solution: False. Consider the graph $v \leftarrow s \leftrightarrow u$ (which clearly has a path from u to v), with DFS starting from s and visiting u as s's first neighbor. Since u's only neighbor s is already black, u finishes immediately, so we "pop back up" to s, and finally visit its other neighbor v. In this execution, v was not discovered until after u was finished.

2. (10 points) Recall the path-relaxation property: if we relax the edges of a particular shortest path $s \overset{p}{\rightsquigarrow} v$ in order (possibly interleaved with arbitrary other relaxations), then $v.d = \delta(s, v)$ afterward.

   Suppose we perform a sequence of relaxations in which the edges of $s \overset{p}{\rightsquigarrow} v$ are *not* relaxed in order. (For example, perhaps one of p's edges is never relaxed at all.) Is it still possible that $v.d = \delta(s, v)$ afterward? If so, give an example graph and sequence of relaxations demonstrating this. If not, argue why not.

   Solution: It *is* possible that this can happen. The key is that there may be more than one different shortest path from s to v, say p and p'. If we relax the edges of p' in order, but not the edges of p, then we are still guaranteed that $v.d = \delta(s, v)$. For example, consider the graph with $w(s, v) = 2$, $w(s, u) = 1$, $w(u, v) = 1$, where p is $s \to v$ and p' is $s \to u \to v$ (which both have weight 2).

3. (10 points) Suppose you are given a weighted, directed graph $G = (V, E)$ where all the edge weights are integers between 1 and 10. Describe a single-source shortest paths algorithm that runs in time $O(V + E)$, and argue why it has this runtime. *Hint:* first introduce some new vertices and edges to the graph.

   Solution: Ideally, to solve this problem we could use an algorithm that we have already studied. The only single-source shortest paths algorithms we studied that run in time $O(V + E)$ are DAG-SSSP for a weighted DAG, and BFS on an unweighted graph. Since our graph may not be a DAG, it doesn't appear that we can use the former algorithm. And since our graph is weighted, we cannot run BFS on it *as is*. But we can *convert* our graph into an *unweighted* one, while preserving path lengths, by replacing each weighted edge of weight w by a sequence of w unweighted edges through new intermediate vertices.

   In more detail, we convert the weighted graph G into an unweighted graph $G' = (V', E')$ as follows. For each vertex in V we create a corresponding vertex in V'. For each edge $(u, v) \in E$ having integer weight w (between 1 and 10), we create $w - 1$ *additional* vertices $u_1, u_2, \ldots, u_{w-1}$ in V', and w *unweighted* edges $(u, u_1), (u_1, u_2), \ldots, (u_{w-1}, v)$ in E'. The distance from u to v in G' is therefore still w, the same as the distance in G, and every path in G' between two nodes of the original graph corresponds exactly to a path in G of the same weight. Therefore, BFS on G' with the original starting vertex returns the single-source shortest paths for the vertices of the original graph G (and for the other new nodes as well, but we can ignore that information).

   We analyze the runtime: in our conversion, for each edge of the original graph we add up to 9 new vertices in V' and create up to 10 edges in E'. So, $|V'| \leq |V| + 9|E|$ and $|E'| \leq 10|E|$. So the running time to construct G' and run BFS on it is bounded by $O(V' + E') = O((V + 9E) + 10E) = O(V + E)$.

4. Recall that a *cycle* in a graph $G = (V, E)$ is a path $v \rightsquigarrow v$ having one or more edges, for some vertex $v \in V$.

   (a) (5 points) Argue the following: for any given $v \in V$, a cycle $v \rightsquigarrow v$ of minimum weight must be of the form $v \to w \overset{p}{\rightsquigarrow} v$ for some $w \in V$ (possibly $w = v$), where $(v, w) \in E$ and $w \overset{p}{\rightsquigarrow} v$ is a shortest path from w to v.

   Solution: Let $v \rightsquigarrow v$ denote a minimum-weight cycle. Since a cycle has at least one edge by definition, let the first edge of that cycle be $v \to w$. Now, since the cycle has to return to v, it must be that the rest of the cycle must be a path $w \overset{p}{\rightsquigarrow} v$ (note that if $w = v$, then this is an empty path). We claim that $w \overset{p}{\rightsquigarrow} v$ must be a *shortest* path from w to v (which is what was to be shown). For if not, then there is a strictly shorter path $w \overset{q}{\rightsquigarrow} v$. But then the weight of the cycle $v \to w \overset{q}{\rightsquigarrow} v$ is strictly smaller than the weight of the original cycle $v \to w \overset{p}{\rightsquigarrow} v$, which contradicts the assumed minimality of the latter cycle.

   (b) (10 points) The *minimum-weight cycle problem* is: given a weighted directed graph $G = (V, E)$ having no negative-weight cycles, output a cycle having minimum total weight, or "no cycle" if none exists.

   Describe an efficient algorithm for the min-weight cycle problem, briefly argue why your algorithm is correct, and state its asymptotic runtime. You will get more points for a faster runtime. *Hint:* build on ideas/algorithms from class!

   Solution: We know by the previous part that a minimum-weight cycle is some edge $v \to w$ followed by a shortest path $w \rightsquigarrow v$. So, our algorithm will consider *all* edges $(v, w) \in E$, along with the shortest path length from w to v. The latter information can be computed for all w, v at once using an all-pairs shortest paths algorithm, such as Floyd-Warshall.

   In detail, the algorithm is as follows:

   1. Run the Floyd-Warshall algorithm on G to compute all-pairs shortest path distances $D[u, v]$ and shortest-path predecessors $\Pi[u, v]$.

   2. For each edge $(v, w) \in E$, compute $w(v, w) + D[w, v]$, which by the previous part is the total weight of a shortest cycle of the form $v \to w \rightsquigarrow v$.

   3. For the *smallest* total weight $w(v, w) + D[w, v]$, output the cycle $v \to w \rightsquigarrow v$, where $w \rightsquigarrow v$ is a shortest path from w to v found using the predecessor information Π output by Floyd-Warshall.

   Note that the for loop runs in time $O(|E|)$, since each lookup of $w(v, w)$ and $D[w, v]$ takes constant time (if we go over the edges in order according to the adjacency lists). The runtime of Floyd-Warshall itself is $O(|V|^3)$. Thus, the total runtime of the algorithm is $O(|V|^3 + |E|) = O(|V|^3)$, since $|E| \leq |V|^2$.

Scrap paper — no exam questions here.