CS/ECE 374 A (Spring 2022)
Homework 9 Solutions

Problem 9.1: We are given a weighted DAG (directed acyclic graph) G with n vertices and m edges with $m \geq n$, where each edge weight may be positive or negative (you may assume that no edge has weight zero). We are also given two vertices $s, t \in V$.

(a) (35 points) Describe an efficient algorithm to determine whether there exists a path from s to t such that the number of positive-weight edges is strictly more than the number of negative-weight edges in the path.

[Hint: there is an $O(m+n)$-time solution (but some partial credit will still be given for an $O(mn)$-time solution). One approach is to use dynamic programming, but a simpler approach is to just run a known algorithm from class on a new weighted graph.]

(b) (65 points) Describe an efficient algorithm for determining whether there exists a path from s to t such that the number of positive-weight edges is strictly more than the number of negative-weight edges in the path *and* the total weight of the path is negative.

[Hint: there is an $O(mn)$-time solution. One approach is to use dynamic programming; another approach is to run a known algorithm on a new graph.]

Solution:

(a) Define a new weighted DAG G' as follows:

- The vertices and the edges are the same as the given DAG G.
- For each edge $(u, v) \in E$, if (u, v) has positive weight in G, define the weight of (u, v) in G' to be -1, otherwise define the weight of (u, v) in G' to be 1.

We compute the shortest path from s to t in G'. The answer is yes iff the shortest path has strictly negative weight.

Justification. The weight of any path in G' is equal to the number of negative-weight edges minus the number of positive-weight edges along the corresponding path in G. Thus, the shortest path weight in G' is strictly negative iff there is a path where the number of negative-weight edges is strictly less than the number of positive-weight edges.

Run time analysis. The graph G' has n vertices and m edges, and can obviously be constructed in $O(m+n)$ time. As explained in class, there is a single-source shortest path algorithm for DAGs which works even when there are negative weights and run in $O(m+n)$ time.

(b) Let $G = (V, E)$ be the given weighted DAG. Define a new weighted DAG G' as follows:

- For each vertex $v \in V$ and for each number $i \in \{-(n-1), \ldots, n-1\}$, create a vertex (v, i) in G'.
- For each edge $(u, v) \in E$ with weight $w(u, v)$ and for each number $i \in \{-(n-1), \ldots, n+1\}$, create an edge $((u, i), (v, i+1))$ in G' with weight $w(u, v)$ if $w(u, v) > 0$, and an edge $((u, i), (v, i-1))$ in G' with weight $w(u, v)$ if $w(u, v) < 0$.
- Create a new vertex t' in G' and add an edge from (t, i) to t' of weight 0 for every $i \in \{1, \ldots, n-1\}$.

We compute the shortest path from $(s, 0)$ to t' in G'. We return true iff the shortest path has strictly negative weight.

Justification. A path π from s to u in G corresponds to a path from $(s, 0)$ to (u, i) of the same total weight in G', where i equals the number of positive-weight edges minus the number of negative-weight edges along π in G (this can be formally proved by induction). Thus, the shortest path from $(s, 0)$ to t' in G' corresponds to the shortest path from s to t in G such that the number of positive-weight edges is strictly greater than the number of negative-weight edges. This shortest path has negative weight iff there exists a path from s to t in G such that the number of positive-weight edges is strictly greater than the number of negative-weight edges and the total weight of the path is negative.

Run time analysis. The graph G' has $N = O(n^2)$ vertices and $M = O(mn)$ edges, and can be constructed in $O(mn)$ time. As explained in class, there is a single-source shortest paths algorithm for DAGs that run in $O(M + N) = O(n^2 + mn) = O(mn)$ time.

(*Note.* Alternatively, the extra vertex t' could be avoided since the DAG shortest path algorithm computes the shortest paths from $(s, 0)$ to all (u, i).)

Problem 9.2: We are given a weighted directed graph $G = (V, E)$ with n vertices, where all edge weights are positive. Each edge is colored red or blue. We are also given an integer $k \leq n$.

We want to compute the shortest closed walk that *contains at least one blue edge and does not have k consecutive red edges.* Describe an efficient algorithm to solve this problem.

(For example, if $k = 4$, a walk with color sequence blue-red-red-blue-red-red-red-blue-blue-red-red-blue is allowed, but not blue-red-red-blue-red-red-red-red-blue. For motivation, imagine that traveling along blue edges lets you recharge. We don't want to travel too long without using a blue edge.)

[Hint: it might be helpful to solve the following all-pairs variant of the problem first: for every pair $u, v \in V$, find the shortest walk from u to v that does not have k consecutive red edges. One approach is to define a new graph and run a known algorithm on the graph.]

[Note: a correct solution with $O(k^2 n^3)$ time will get you 90 points; a correct solution with $O(kn^3 \log n)$ or $O(kn^3)$ time will get you 100 points (full credit); and a solution with $O(n^3 \log n)$ time or better will receive 15 more bonus points!]

Solution: Let $G = (V, E)$ be the given weighted directed graph with n vertices and $m \leq n^2$ edges. Let $w(u, v)$ denote the weight of the edge (u, v) in G. Define a new weighted directed graph G' as follows:

- For each $v \in V$ and $i \in \{0, \ldots, k-1\}$, create a vertex (v, i) in G'.
- For each red edge $(u, v) \in E$ and each $i \in \{0, \ldots, k-2\}$, create an edge $((u, i), (v, i+1))$ in G' with weight $w(u, v)$.
- For each blue edge $(u, v) \in E$ and each $i \in \{0, \ldots, k-1\}$, create an edge $((u, i), (v, 0))$ in G' with weight $w(u, v)$.

For every $u \in V$, find the shortest path weight $d_{G'}((u, 0), (v, i))$ from $(u, 0)$ to all vertices (v, i) in G'.

We find the blue edge $(u^*, v^*) \in G$ and an index $i^* \in \{0, \ldots, k-1\}$ that minimizes $d_{G'}((v^*, 0), (u^*, i^*)) + w(u^*, v^*)$. We return the closed walk formed by concatenating the edge (u^*, v^*) with the walk from v^* to u^* that corresponds to the shortest path from $(v^*, 0)$ to (u^*, i^*) in G'.

Justification. A walk from $(u, 0)$ to (v, i) in G' corresponds to a walk from u to v in G that does not have k consecutive red edges and ends with i consecutive red edges, for any $i \leq k-1$ (this can be formally proved by induction). We want a shortest closed walk that contains a blue edge and does not contain k consecutive red edges. We can guess a blue edge (u^*, v^*) in the solution, and the rest of the walk must then be a shortest walk from v^* to u^* without k consecutive red edges, which has weight $\min_{i^*} d_{G'}((v^*, 0), (u^*, i^*))$.

Run time analysis. The graph G' has $N = O(kn)$ vertices and $M = O(km)$ edges. For each $u \in V$, all shortest paths from $(u, 0)$ can be found by running Dijkstra's single-source shortest paths algorithm, which takes $O(N \log N + M) = O(kn \log n + km)$ (since $\log(kn) \leq \log(n^2) = O(\log n)$). Since we run Dijkstra's algorithm n times (one for each $(u, 0)$), the total time is $O(kn^2 \log n + kmn)$.

In the final step of computing u^*, v^*, i^*, we loop through $O(n^2)$ choices for (u^*, v^*) and k choices for i^*. This step takes $O(kn^2)$ additional time.

Overall run time: $O(kn^2 \log n + kmn)$, which is at most $O(kn^3)$.

(*Note.* if instead of Dijkstra's algorithm we run Floyd and Warshall's all-pairs shortest paths algorithm, the running time would be $O(N^3) = O(k^3 n^3)$, which is slower. If we run Dijkstra's algorithm from all sources, the running time would be $O(N \cdot (N \log N + M)) = O(k^2 n^2 \log n + k^2 mn) \leq O(k^2 n^3)$, which is also slower.)

Sketch of a Better Solution (worth bonus points): Define $R(u, v, \ell)$ to be the weight of the shortest path from u to v that uses only red edges and have length at most ℓ.

First stage. We first use dynamic programming to compute $R(u, v, k-1)$ for all $u, v \in V$. (This part is similar to the "repeated squaring" method for APSP, on the subgraph formed by the red edges.) For the base case, we have $R(u, v, 0) = \infty$ if $u \neq v$, and $R(u, v, 0) = 0$ if $u = v$, for each $u, v \in V$. For the recursive formula, we have

$$R(u, v, \ell) = \begin{cases} \min\limits_{x \in V} (R(u, x, \ell/2) + R(x, v, \ell/2)) & \text{if } \ell \text{ is even} \\ \min\limits_{x \in V:\ (x,v) \text{ is a red edge}} (R(u, x, \ell-1) + w(x, v)) & \text{if } \ell \text{ is odd} \end{cases}$$

for each $u, v \in V$ and each $\ell \geq 1$.

Observe that in computing $R(u, v, k-1)$, we only need to generate $R(\cdot, \cdot, \ell)$ for a sequence of $O(\log k)$ many ℓ's. Thus, the number of subproblems needed is $O(n^2 \log k)$, and each subproblem takes $O(n)$ time. The total time of the first stage is $O(n^3 \log k)$.

Second stage. We now solve the original problem. Define a new weighted directed graph G' as follows:

- For each $v \in V$, create vertices $(v, 0)$ and $(v, 1)$ in G'.
- For each $u, v \in V$, create an edge $((u, 0), (v, 1))$ in G' with weight $R(u, v, k-1)$. (Note that in particular, there will be an edge from $((u, 0), (u, 1))$ with weight 0.)
- For each blue edge $(u, v) \in E$, create an edge $((u, 1), (v, 0))$ in G' with weight $w(u, v)$.

Note that G' has $2n$ vertices and $O(n^2)$ edges, and can be constructed in $O(n^2)$ time using the $R(\cdot, \cdot, k-1)$ values computed from the first stage.

We run Floyd and Warshall's all-pairs shortest paths algorithm on G'. This takes $O(n^3)$ time.

To finish, we find the blue edge $(u^*, v^*) \in G$ that minimizes $d_{G'}((v^*, 0), (u^*, 1)) + w(u^*, v^*)$. This takes $O(n^2)$ additional time.

The total run time over both stages is $O(n^3 \log k)$.