CS2800 Design and Analysis of Algorithms
Problem Set #5 February 27, 2023

1. Given a graph G and a source vertex s, modify the Bellman-Ford algorithm to check if there is a negative weight cycle reachable from s, and if so, find one such cycle.

Solution: Let $\text{dist}^i(v)$ denote the value of $\text{dist}(v)$ after the i^{th} iteration of the main loop of the Bellman-Ford algorithm. What we saw in class is that $\text{dist}^i(v) \leq \text{dist}_{\leq i}(v)$. If $\exists v$ such that $\text{dist}^n(v) < \text{dist}^{n-1}(v)$, then there must exist a negative weight cycle in the graph. We want to show that a negative weight cycle will exist in the shortest walk from s to this particular v.

First, observe that if $\text{dist}^n(v) < \text{dist}^{n-1}(v)$, then the path with distance $\text{dist}_{\leq n}(v)$ must actually contain n edges (prove this). This means that some vertex repeats in the path from s to v. Now, modify the Bellman-Ford algorithm to keep the parent pointers each time an edge is relaxed. Use these parent pointers to trace the path from s to v and you will find the cycle.

2. Given a graph G and two vertices s and t, the replacement path P_e is the shortest path from s to t in $G - e$. Suppose that the shortest path in G from s to t passes through every vertex in G, give an $O(E \log V)$-time algorithm to compute the distance of replacement paths P_e for every edge e. Assume that the graph has no negative weight edges.

Solution: Let $P = s = u_0 - u_1 - u_2 - \cdots - u_{n-1} = t$ be the shortest path from s to t in G. You can find this path in $O(E \log V)$ time. For every edge, $e \notin P$, the shortest path does not change. So, we need to find the P_e for all $e = (u_i, u_{i+1})$.

The straightforward way is to run Dijkstra's algorithm for each such e that is removed. This would give a running time of $O(VE \log V)$. For a faster algorithm we proceed as follows.

For an edge $e = (u_i, u_{i+1})$, let $V_i = \{s, u_1, \ldots, u_i\}$ and $V_i' = V - V_i$. Let $E_i = \{(u, v) | u \in V_i, v \in V_i'\} - \{e\}$.

Verify the following lemma.

Lemma. $\sum_{e' \in P_e} w(e') = \min_{e'=(u,v) \in E_i} \{d(s, u) + w(u, v) + d(v, t)\}$.

Use priority queues carefully to obtain the value of P_e from P_e' where $e = (u_i, u_{i+1})$ and $e' = (u_{i-1}, u_i)$.

3. For a weighted graph G and two vertices s and t, the *best path* between s and t is the path with minimum weight that contains the least number of edges. Assuming that the weights of

the edges in the graph are non-negative, modify Dijkstra's algorithm to obtain the best paths from s to all the other vertices in the graph.

Solution: Run Dijkstra's algorithm taught in class on G. But, now for each vertex store the distance from s as well as the number of edges in the path that gives that distance. The comparison step will now first compare the distances, and if they are the same, will compare the number of edges in the path. Running time will be same as Dijkstra's algorithm.

4. Given a directed graph G with non-negative edge weights, and two vertices s and t, design an algorithm to find the shortest *walk* from s to t (shortest in terms of sums of weights of edges in the walk) such that the number of edges in the walk is a multiple of 3.

   A *walk*, as opposed to a path, is a sequence of moves in the graph where vertices can repeat.

   Solution: Given $G(V, E)$, construct a graph $G'(V', E')$ where for each $v \in V$, we have three vertices $(v, 0)$, $(v, 1)$, and $(v, 2)$ in V' and for each edge $(u, v) \in E$ add edges (u, i) to $(u, (i + 1) \bmod 3)$.

   Now every walk in G corresponds to a walk in G', and every walk in G' corresponds to a walk in G (prove this). Also, the shortest path (*without repeating vertices*) from $(s, 0)$ to $(t, 0)$ in G' will correspond to a walk from s to t in G of shortest length whose number of edges is a multiple of 3 (prove this).

   Now, run Dijkstra's algorithm on G', and this has a running time of $O((V+E) \log V)$ since $E' = O(E)$ and $V' = O(V)$.

5. Consider a weighted digraph $G(V, E)$ whose edge weights change over time. You can think of your digraph as representing the road network of a city where an edge $e = (u, v)$ is the road connecting two junctions u and v. Naturally, the time to traverse that road depends on the traffic, and this varies over time.

   To model this situation, assume that for every edge e there is a function $f_e : \mathbb{N} \to \mathbb{N}$ (which is its weight that varies over time) such that the following conditions hold:

   - You don't own a time machine - when crossing $e = (u, v)$, you cannot reach v at a time before the time at which you were at u: More formally,

     $$\forall t, f_e(t) > 0.$$

   - You don't overtake within city limits - while crossing $e = (u, v)$, if your friend reaches u earlier than you, then you cannot reach v earlier than your friend. More formally,

     $$t_1 \leq t_2 \implies t_1 + f_e(t_1) \leq t_2 + f_e(t_2).$$

Given such a digraph $G(V, E, f)$, and two vertices s and t, design an algorithm to find the path that takes shortest time to commute from s to t. Assume that you start from s at $t = 0$, and you don't care about the number of roads you take, but just the total commute time. Furthermore, assume that given t, you can compute $f_e(t)$ in $O(1)$-time for every $e \in E$.

Solution: Same as Dijkstra. Just prove the correctness imitating the proof that was done in class.

6. Consider the greedy algorithm for the interval scheduling problem that we saw in class. Let $\mathcal{I}$ be an instance of the problem and let OPT be the maximum number of non-overlapping intervals in $\mathcal{I}$. Let k be the number of intervals returned by the shortest-interval first algorithm. Show that $k \geq \text{OPT}/2$.

Solution: Let I be an interval chosen by the shortest-interval first algorithm. It can overlap with at most two intervals from OPT. No interval from OPT is fully contained in I, because then this interval would have been chosen by the shortest-interval first instead of I. Hence for any interval intersecting I must have its start time before the start time of I or its end-time before the end-time of I. There can be at most two such intervals in OPT since OPT itself is a set of non-overlapping intervals.

Therefore, $\text{OPT} \leq 2k$.