CS 341: Assignment 6

x969li

Q2. [10 marks] Monotonically Increasing Shortest Path.

Input: A weighted directed graph $G = (V, E; w)$ with n vertices and m edges. A vertex $s \in V$.

Description of Algorithm:

Use the same logic with that of the Bellman-Ford Algorithm.
Let $D[1...n]$ be an array where $D_i[v] = (w_e, w)$ has the weight of the monotonically increasing shortest path P from s to v at the end of i-th iteration, with w_e be the weight of the edge $(u, v) \in P$ (last edge in P), for some $u \in V$, and w is the weight of P.
Use an array $P[1...n]$ to store the information of the paths from s to v that might be a part of the shortest path for some vertices for each vertex v.

To improve the performance (or efficiecy), we can create a linkedlist for $P[v]$ for all $v \in V$. A node has fields: `e` , `w` , and `next` , where `e` is the weight of the last edge in the path represented by this node, and `w` is the total weight of this path.
We use a 2-D array $\texttt{check}[1...n, 1...n]$ to indicate the nodes (paths) that was added in i-th round but haven't been checked, for some $1 \leq i \leq n - 1$.
For example: Notice that $\texttt{check}[u, v]$ is some path from s to vertex u.
If $\texttt{check}[u, v] = f$, then when iterating over $(u, v) \in E$, we should check all the paths between the node f.next and $P[u]$ inclusively, then decide whether adding the edge (u, v) to this path and update $P[v]$, or even $D[v]$.

Initialization: $D[v] := (-\infty, \infty)$ for all $v \in V$, $D[s] := (-\infty, 0)$, $P[v] := (\infty, \infty)$ for all $v \in V$, $P[s]$.next := Node$(-\infty, 0)$, $P[s] := P[s]$.next.
When checking $(u, v) \in E$, iterate through all paths between $\texttt{check}(u, v)$.next and $P[u]$ with last-edge-weight and total-weight to be w_e and w.
If $w_{(u,v)} > w_e$, there are two cases:

1. If $w + w_{(u,v)} < D[v].w$, we add $(w_{(u,v)}, w + w_{(u,v)})$ to the end of $P[v]$. Also, we assign $D[v]$ with $(w_{(u,v)}, w + w_{(u,v)})$.
2. Otherwise, if $w_{(u,v)} < D[v].w_e$, then add $(w_{(u,v)}, w + w_{(u,v)})$ to the end of $P[v]$.

We need to set $P[v] := P[v]$.next whenever the linked list that $P[v]$ belongs to is updated.
This algorithm updates $\texttt{check}(u, v) := P[u]$ after the loop.

Pseudocode

$\texttt{shortest-path}(G = (V, E; w), s)$

1. $D[v] := (-\infty, \infty)$ for all $v \in V$ // Denote $D[v]$.first as $D[v].e$, $D[v]$.second as $D[v].w$.
2. $D[s] := (-\infty, 0)$
3. $P[v] := \texttt{Node}(\infty, \infty)$ for all $v \in V$ // Node(e, w) creates a Node which represents a path of total weight w, while its last edge's weight is e. The `next` field is set to **null** by default.
4. $\texttt{check}[i, j] := P[i]$ for all $1 \leq i, j \leq n$
5. $P[s]$.next $:= \texttt{Node}(-\infty, 0)$
6. $P[s] := P[s]$.next
7. **for** i **from** 1 **to** $n - 1$ **do**
8.   **for each edge** $(u, v) \in E$ **do**
9. $nde := \texttt{check}[u, v]$.next
10.   **while** nde **is not null do**
11.   **if** $nde.e < w_{(u,v)}$ **then**
12.   **if** $nde.w + w_{(u,v)} < D[v].w$ **then**
13. $P[v]$.next $:= \texttt{Node}(w_{(u,v)}, nde.w + w_{(u,v)})$
14. $P[v] := P[v]$.next
15. $D[v] := (w_{(u,v)}, nde.w + w_{(u,v)})$
16.   **else if** $w_{(u,v)} < D[v].e$ **then**
17. $P[v]$.next $:= \texttt{Node}(w_{(u,v)}, nde.w + w_{(u,v)})$
18. $P[v] := P[v]$.next
19. $nde := nde$.next
20. $\texttt{check}[u, v] := P[u]$
21. **return** $D[v].w$ for all $v \in V$

Run-time Analysis:

Line 1, 3 takes $O(n)$ time, whereas line 2, 5, 6 takes $O(1)$ time. Line 4 takes $O(n^2)$ time.

The runtime of the for-loop from line 8 to 20 must be $O(nm)$.
Since there are at most $n - 1$ vertices u such that $(u, v) \in E$, on average $P[v]$ grows $O(n)$ in each iteration, i.e. the length of the paths from s to u which are unchecked for the shortest path from s to v is $O(n)$ on average in the worst case.
Therefore, the runtime of the code from line 8 to line 20 is $O(nm)$ since there are m edges.
Hence, the code from line 7-20 has runtime $O(n^2m)$.

Therefore, this algorithm has running time $O(n^2m)$.

Proof of Correctness:

Similar to the Bellman-Ford Algorithm, to obtain the shortest path for every vertex, at most $n-1$ edges will be used. Thus, the for loop in line 7 is correct.
Also, we need to check every edges in each round to see if they would contribute to the shortest path for some vertex, according to the Bellman-Ford algorithm. (Line 8)
At the end of i-th iteration, for all $v \in V$, the linkedlist of which end node is $P[v]$ contains all the paths from s to v using $\leq i$ edges which might be a part of some monotonically increasing shortest path of some vertices $\in V$. It might has more paths than that. This invariant is maintained by line 10-20, which will be explained in the following.

For each edge $(u,v) \in E$, since $\texttt{check}[u,v]$.next is the start of the nodes (paths) in the linked list that are not checked by vertex v, we check whether (u,v) can be added at the end of the path P for all P in the linked list from $\texttt{check}[u,v]$.next to $P[u]$, where $P[u]$ is the end of that linkedlist.
If $P.e$, which is the last edge of the path P, has weight $\geq w_{(u,v)}$, then (u,v) cannot be added to that path. So we only consider the otherwise situation. (Line 11)
If $P+(u,v)$ has weight smaller than $D[v].w$, then we update $D[v]$. This allows all vertices to obtain the monotonically increasing shortest path better than or the same with the one that can be obtained using $\leq i$ edges at the end of i-th looping. (Line 15)
Line 13-14 and Line 16-18 adds all path from s to v that could be a part of the shortest path from s to some vertices t to the linkedlist of which end node is $P[v]$.
If $P+(u,v)$ has weight $\geq D[v].w$ and $w_{(u,v)} \geq D[v].e$, then $P+(u,v)$ won't be in any shortest path, because choosing $D[v]$ over $P+(u,v)$ will allow smaller overall weights while allowing the path to be monotonically increasing if any path that contains $P+(u,v)$ is monotonically increasing. (Line 16)
Therefore, the pseudocode from line 8 to 20 maintains the following invariants:
For all $v \in V$,

1. $D_i[v]$ stores the information of the shortest path from s to v that is better than or the same with the one that can be obtained using $\leq i$ edges.
2. $P_i[v]$ is the end of the linkedlist which includes but is not limited to all the paths from s to v using $\leq i$ edges which might be a part of some monotonically increasing shortest path of some vertices $\in V$.
3. For all $u \in V$, at the end of i-th round, if it is not null, $\texttt{check}_i[u,v]$.next is the start of the nodes that are unchecked by vertex v in the linkedlist ended by $P[u]$ —— the linkedlist for the paths from s to u.

Hence, all in all, this algorithm is correct because at the end of the $(n-1)$-th iteration, we can find the monotonically increasing shortest path from s to v for all $v \in V$ according to the invariants.