Decremental algorithm for maintaining Shortest-Path Tree

Problem 1 (Decremental shortest-path tree)**.** *Let $G = (V, E)$ be a directed graph initially with n vertices and m edges, and let s be a source. There is an online sequence $\sigma = (e_1, e_2, \ldots, e_m)$ of m edge deletions. At time $t = 1$, edge e_1 is removed from G; at time $t = 2$, edge e_2 is removed from G, an so on.*

The goal is to obtain a decremental algorithm for maintaining the shortest-path tree of G rooted at s, in an efficient manner.

Remark The sequence σ is NOT known to us beforehand and we get to know a deleted edge only at the time when it is removed from G.

1 Initialization Phase:

Let T be initialized to a shortest-path tree of G rooted at s. For each vertex $v \in V$, we store in $dist(v)$ the distance from s to v in G, and we store in $parent(v)$ the parent of v in the tree T. If $dist(v) = \infty$, then $parent(v)$ is NULL. For each $v \in V$, we set A_v, N_v to $\{x \mid (x, v) \in E\}$. Throughout our algorithm, A_v will be a subset of N_v, and N_v will be the set of all in-neighbours of v in G.

2 Steps to update T on an edge deletion:

Let $e = (u, v)$ be an edge that is deleted from G.

Algorithm 1: Delete $e = (u, v)$

```
Update A_v = A_v − {u} and N_v = N_v − {u}, also remove e to E;
if u = parent(v) then
    for i ∈ [1, n − 1] do Set L_i = {w ∈ T(v) | dist(w) = i};
    for w ∈ T(v) do Set dist(w) = ∞;
    for i ∈ [1, n − 1] do
        for y ∈ L_i do
            for x ∈ A_y do
                if dist(x) ≥ i then Remove x from A_y ;
                else
                    Set parent(y) = x and dist(y) = i;
                    Break the loop;
            if A_y = ∅ then
                Move y to L_{i+1}, and re-set A_y to N_y;
```

3 Correctness Analysis:

Lemma 1. *During the period when $y \in V$ is at level i, the set A_y represent the in-neighbors of y that have not been scanned yet for y at level i.*

Lemma 2. *After execution of Algorithm 1, the set L_i comprises of the vertices whose distance from s in the updated graph is i.*

Lemma 3. *Algorithm 1 correctly updates T.*

4 Time Analysis:

For a vertex y at a given level, each incoming-edge (x, y) is scanned only once. As the level y (or the distance from s to y decreases at most n times, for the vertex y we are spending $O(deg(y) \cdot n)$ time, where $deg(y)$ is the degree of y in the starting graph. Summing over all vertices gives us that the total complexity is $O(mn)$. So amortized update time (average update time per edge) is $O(n)$.