Incremental algorithm for maintaining Shortest-Path Tree

Problem 1 (Incremental shortest-path tree)**.** *Let $G = (V, E = \emptyset)$ be a directed graph initially with n vertices and 0 edges, and let s be a source. There is an online sequence $\sigma = (e_1, e_2, \ldots, e_m)$ of m edge insertions to G. At time $t = 1$, edge e_1 is added to G; at time $t = 2$, edge e_2 is added to G, an so on.*

The goal is to obtain an incremental algorithm for explicitly 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 an inserted edge only at the time when it is added to G.

1 Initialization Phase:

We explain the initialization phase for the general scenario when edge-set E could be non-empty.

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.

We also initialize the list of out-neighbours $N(v) = \{x \mid (v, x) \in E\}$, for each $v \in V$.

2 Steps to update T on an edge insertion:

Let $e = (x, y)$ be an edge that is added to G. The algorithm to update T on addition of edge e is below.

Algorithm 1: Insert $e = (x, y)$

```
1  Add y to N(x), and e to E;
2  if dist(y) ≳ dist(x) + 1 then
3      Set parent(y) = x;
4      Set i_0, dist(y) = dist(x) + 1 ;
5      Set L_{i_0} = {y};
6      for i ∈ {i_0, i_0 + 1, ..., n − 1} do
7          Set L_{i+1} = ∅;
8          foreach a ∈ L_i and b ∈ N(a) do
9              if dist(b) ≳ dist(a) + 1 then Set parent(b) = a, dist(b) = dist(a) + 1, and add b to
                 L_{i+1};
10         end
11     end
12 end
```

Lemma 1. *L_i represent the set of those vertices v whose distance ($dist(v)$) in the new graph has become i.*

Lemma 2. *The set $V_e := \bigcup_{i \geq i_0} L_i$ represent all vertices whose level is decreased on addition of edge e.*

Lemma 3. *The Algorithm 1 correctly updates T.*

Hint: We are mimicking the algorithm to compute BFS tree of G rooted at y, however we only visit the vertices in set $V_e = \bigcup_{i \geq i_0} L_i$ whose distance from s decreases on addition of edge e.

3 Time Analysis:

Let $V_e = \bigcup_{i \geq i_0} L_i$ be set of all vertices whose level is decreased on addition of edge e.

Then time to update T after addition of e is $O(n) + \sum_{v \in V_e} |N(v)|$.

Note that a vertex is added to V_e if and only if its level changes in T. This can happen at most n times.

Thus, time spent in scanning list $N(v)$ (perhaps multiple times, see step 8), during entire run of algorithm is $O(n \cdot |N(v)|)$.

This directly shows that the total time to update G is $O(mn)$.

As there are m edge insertions, the average-time (amortized-time) spent per edge insertion is $O(n)$.

Remark In this article, we only discussed the incremental scenario. The steps and analysis can be modified to obtain a decremental algorithm as well for maintaining a shortest path tree of a graph undergoing edge deletions.