Lightest Paths in Directed Acyclic Graphs

We discussed an algorithm for Single Source Lightest paths in DAGS (This algorithm is discussed in Section 24.2 of Cormen).

To start we construct a topological sort of the graph: $\{x_0, ...x_{n-1}\}$ with $s = x_0$. It had the following structure:

Algorithm DAG Shortest Paths

$d[x_0] := 0$
For $i = 1$ to $n-1$ do $d[x_i] := \infty$.
For $i = 1$ to $n-1$ do
  For each edge $x_j x_i$ into x_i do
    If $d(x_j) + w(x_j, x_i) < d(x_i)$ Then
      $d[x_i] := d[x_j] + w(x_i, x_j)$
      $p[x_i] := x_j$

To certify that our algorithm works we need to show that properties (1)-(4) hold at the end of the algorithm (these properties are listed in the Certifying a Lightest Path Tree handout).

We set $d(s) = 0$ and never change it so (4) will hold at the end of the procedure. Since G has no cycles, we see that following the predeccor pointers from v yields the (reverse of the) path fron s to v in the graph of (3), so this graph must indeed be a rooted tree (Recall that a directed graph is a rooted tree rooted at s precisely if every node is reachable from s and the number of arcs is exactly one less than the number of vertices).

We prove inductively that the inequalites in (1) and (2) hold for all $x_k x_l$ with $k < l \leq i-1$ at the beginning of iteration i. Clearly this is true when $i = 1$ as there are no such arcs. Note that in iteration i, the only d or p value we change is that for x_i. So the inequalities in (1) and (2) continue to hold for $x_k x_l$ with $l \leq i-1$ and we need only check these properties for the arcs $x_k x_i$. Now, the inequality of (1) holds for such arcs as otherwise we would update the value of $d[i]$ when we considered the arc $x_k x_i$. Also, at the beginning of Iteration i, $d[x_i]$ is still ∞ and $p[x_i]$ is still undefined so if we update $p[x_i]$ we also update $d[x_i]$ so that if $x_i \in V'$ then inequality of (2) holds for the arc $p[x_i]x_i$. This completes our proof that the algoirthm is correct.

This algorithm is easy to implement in $O(|E|)$ time provided we have a reverse adjacency list of G, that is a list of all the arcs into each vertex. This structure is easy to build in $O(|E|)$ time.