CSC 226 - Assignment 3 - Theory

Daniel Frankcom

March 2017

1. When a smaller tree is attached to a larger tree, the height of the smaller tree at least doubles, since $H_{small} < H_{large}$, meaning that the new height is $2H_{small} + (H_{large} - H_{small})$.
   Since this is the case, we know that there must be a logarithmic upper bound on the overall height of the tree, as the height of our smaller tree may only be doubled at most $logN$ times. This is due to the fact that doubling the height $logN$ times results in N items in our tree, which by definition is the maximum.

2. If Algorithm 4.10 did not compute the correct shortest paths for our graph, then it means that there is at least 1 edge which was not relaxed during the algorithm's operation.
   Since we start with a topological ordering, each edge may be relaxed only once, as there is no way for the distance from the source vertex to our current vertex to increase. This is because we have only previously considered vertices that come prior to our current vertex in the topological ordering, and are therefore closer to the source.
   Since every edge is only considered once, and optimally, it is not possible to have missed an edge relaxation in the process of traversing our graph.
   $\therefore$ by contradiction, the theorem is true.

3. The complexity of Dijkstra's algorithm comes from the following equation:
   O($|E|$*time to decrease key+$|V|$*time to extract minimum)
   Since our PQ is unordered, the time to decrease the keys is not a factor, and is therefore 0, leaving us with simply:
   O(n*time to extract minimum) - where $|V| = n$
   Since our PQ is unordered, the most efficient way that we can find the minimum element is by traversing the entire array. Therefore our final runtime is:
   O($n * n$)=O(n^2)

4. The Bellman-Ford algorithm, when run on a graph with no negative-weight cycles, finds the length of the shortest path from every vertex to every other. If the algorithm is run, and edges can still be relaxed, then it means not that the algorithm has missed an edge relaxation, but instead that this edge will continue to be relaxable to infinity. This is because a negative-weight cycle has no shortest path, as it will continue to decrease as more and more iterations are run. Therefore we can check to see if a graph has a negative-weight cycle by running one more iteration of the algorithm on an already completed list of path distances, and by comparing the 2 results, can determine if such a cycle exists.