Zijian Gao
26319560
2/29/2016
CS311 HW3

22.4-5)

This way of topological sorting can be done using an array to represent in-degrees and list to hold sorted vertices. Assuming that the graphs this will deal with have up to a countably infinite amount of vertices

Initialization:

Create an in-degree array D[1..n] representing the in-degree of each vertex by iterating over the edges, increment in-degree for each incident-to vertex.
Create an output linked list to hold the sorted vertices

Steps:

1. Initialize a queue with all of the 0 in-degree vertices, assuming standard queue routines are available.
2. Loop condition: if there exists an element e in the queue, loop and dequeue. Add this element to the tail of the topological sorted list(such that the first item dequeued is at the front/head of the list)
    1. Decrement all degrees of vertices in adj[e] by 1, effectively removing the edges.
    2. Enqueue any vertices that are now 0 in-degree by using the same iterative method as initializing the array.
    3. Go to top of loop
3. Output the sorted list

Proof of correctness:
Iterating over all edges of the graph allows for all of the incident-to vertices to be discovered trivially. There will be n vertices in D, each with a value representing how many edges enter them.

The queue is then initialized by searching for zero degrees in the degree matrix, properly holding all vertices that must be removed from the graph.

The loop properly terminates, because a DAG with no cycles must have at least one 0 in-degree vertex, which at worst case be the only item in the queue. When the vertex is dequeued, degrees for its adjacent vertices are decremented and more vertices will then have 0 in-degree because this has the same effect of removing a vertex from the DAG therefore it does not add any edges to the graph. As no additional edges are added, the graph is still a DAG, and so there still must exist at least one 0 in-degree vertex after the previous was removed. Since there is a countably infinite amount of vertices, removing a minimum of one vertex at a time will at some point result in an empty graph and properly topologically sorted list.

Because the loop properly terminates and the initial conditions are proper, the algorithm is correct.

If G has cycles, the property that removing a 0 in-degree vertex will always yield at least one additional 0 in-degree vertex on the next iteration is not always true. In the worst case, this algorithm will output an empty list in the case of a graph where each vertex is a part of a cycle such that there is no vertex with 0 in-degree. In a more moderate case, this algorithm will output a list of the vertices that are not part of a cycle, which will not be an accurate topological sorting of the vertices. The concept of topological sorting as defined in this problem does not apply to graphs with cycles.