CS 430 Lecture 28 Activities

Opening Questions

- We saw the Bellman-Ford algorithm found the shortest path from a source to all other vertices by "brute force" every edge in the graph in a fixed order $|V|-1$ times. Why did it need to do this $|V|-1$ times? And with this in mind, could we improve on the Bellman-Ford for certain graphs? At least relax edges leaving from source first. Possible that the shortest path $u \rightsquigarrow v$ goes through every other vertex $|V|-1$ edges might be relaxed in opposite order.

DAG Shortest Path Algorithm

By relaxing the edges of a weighed DAG (directed acyclic graph) $G = (V, E)$ in topological sort order of its vertices, we can compute shortest paths from a single source. Shortest paths are always defined in a DAG, since even if there are negative-weight edges, no negative weight cycles can exist.

Algorithm 28.1 DAG Shortest Path $O(V^2)$

```
function DAG-SHORTEST-PATH(G, w, s)
    topologically sort the vertices of G
    INIT-SINGLE-SOURCE(G, s)
    for all vertex u, taken in topologically sorted order do

        for all vertex v ∈ Adj[u] do
            RELAX(u, v, w)
        end for
    end for
end function
```

1. Here is the topological sort on a DAG. Find the shortest path from s to every other vertex.
2. What is the runtime for DAG Shortest Path? $O(V+E) \Rightarrow O(V+V^2) \Rightarrow O(V^2)$
3. Discuss why DAG Shortest Path is correct.
4. If we restrict the graph to having no negative edges, given a source s, what is the shortest path from s to one of its adjacent vertices?

Dijkstra's Shortest Path Algorithm

- No negative-weight edges.
- Essentially a weighted version of breadth-first search.
  - Instead of a FIFO queue, uses a priority queue.
  - Keys are shortest-path weight estimates ($d[v]$).

Dijkstra's Algorithms
https://www.youtube.com/watch?v=wtdtkJgcYUM https://www.cs.usfca.edu/~galles/visualization/
Dijkstra.html