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$O(V^2)$

```
1: function DAG-Shortest-Path(G, w, s)
2:     topologically sort the vertices of G
3:     Init-Single-Source(G, s)
4:     for all vertex u, taken in topologically sorted order do
5:         for all vertex v ∈ Adj[u] do
6:             Relax(u, v, w)
7:         end for
8:     end for
9: 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. Given a source, you relax an edge, the incremental improvement on source path estimate.
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?

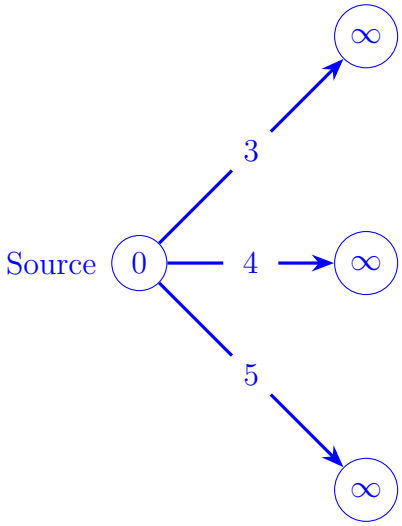


Figure 28.1: 3 is done, because there is no negative path that could go from one of the other nodes to it. 4 and 5 are not done because there could be paths with values that could create a better estimate like $\frac{1}{2}$ and 1 respectively.

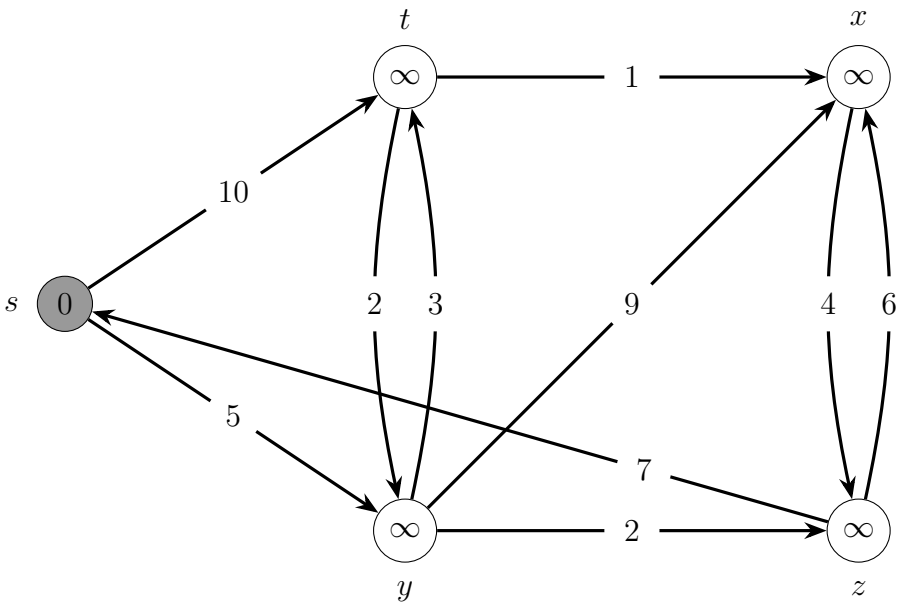
Dijkstra's Shortest Path Algorithm

- 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 ($d[v]$).
- Have two sets of vertices:
  - S = vertices whose final shortest-path weights are determined,
  - Q = priority queue = $V - s$.
- Dijkstra's algorithm can be viewed as greedy, since it always chooses the "lightest" ("closest") vertex in $V - S$ to add to S.

Algorithm 28.2 Dijkstra

```
1: function DIJKSTRA(V, E, w, s)
2:     INIT-SINGLE-SOURCE(V, s)
3:     S ← ∅
4:     Q ← V                          ▷ i.e., insert all vertices into Q by d values
5:     while Q is not empty do
6:         u ← EXTRACT-MIN(Q)
7:         S ← S ∪ {u}
8:         for all vertex v ∈ Adj[u] do
9:             RELAX(u, v, w)    ▷ Possibly updates a short path estimate d value and moves the vertex forward in the queue
10:        end for
11:    end while
12: end function
```

5. Here is a graph with non negative edges. Find the shortest path from s to every other vertex using Dijkstra's algorithm.



6. What is the runtime for Dijkstra's algorithm?

7. Prove the greedy choice in Dijkstra's algorithm (pick the vertex with the smallest shortest path estimate, not including the vertices we are done with) leads to an optimal solution.

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