Shortest Paths—Definitions

- For the next few lectures, we will have a directed graph $G = (V, E)$, and a weight function $w : E \rightarrow \mathbb{R}^{|E|}$.
- We are interested in finding the shortest-path weights from $u$ to $v$, which we will denote $\delta(u, v)$.
- $\delta(u, v) = \infty$ if there is no path from $u$ to $v$ in $G$.
- (Single Source) shortest-path algorithms produce a label: $d[v] = \delta(s, v)$.
- Initially $d[v] = \infty$, reduces as the algorithm goes, so always $d[v] \geq \delta(s, v)$.
- Also produce labels $\pi[v]$, predecessor of $v$ on a shortest path from $s$.

Initializing and Relaxing

**INIT-SINGLE-SOURCE(V, s)**

1. for each $v \in V$
2. do $d[v] \leftarrow \infty$
3. $\pi[v] \leftarrow \text{NIL}$
4. $d[s] \leftarrow 0$

**RELAX(u, v, w)**

1. if $d[v] > d[u] + w_{uv}$
2. then $d[v] \leftarrow d[u] + w_{uv}$
3. $\pi[v] \leftarrow u$
Lemmas

**Lemma**

Any subpath of a shortest path is a shortest path

**Lemma**

Shortest paths can't contain cycles

Path Relaxation Property

Let \( P = \{ v_0, v_1, \ldots v_k \} \) be a shortest path from \( s = v_0 \) to \( v_k \). If the edges \((v_0, v_1), (v_1, v_2), (v_{k-1}, v_k)\) are relaxed in that order, (there can be other relaxations in-between), then \( d[v_k] = \delta(s, v_k) \)

Bellman-Ford Algorithm

- Works with Negative-Weight Edges
- Returns **true** is there are no negative-weight cycles reachable from \( s \), **false** otherwise

**Bellman-Ford**\((V, E, w, s)\)

1. **Init-Single-Source**\((V, s)\)
2. for \( i \leftarrow 1 \) to \(|V| - 1\)
3. do for each \((u, v)\) in \( E\)
4. do **Relax**\((u, v, w)\)
5. for each \((u, v)\) in \( E\)
6. do if \( d[v] > d[u] + w_{uv} \)
7. then return **False**
8. return **True**

Single Source Shortest Path on a DAG

**DAG-Shortest-Paths**\((V, E, s, w)\)

1. **Init-Single-Source**\((V, s)\)
2. topologically sort the vertices (HOW)
3. for each \( u \) in topologically sorted \( V \)
4. do for each \( v \in Adj[u] \)
5. do **Relax**\((u, v, w)\)

SSSP-DAG Analysis and Correctness

- **Correctness**
  - Since vertices are processed in topologically sorted order, edges of any path are relaxed in order of appearance on the path
  - Thus, edges on any shortest path are relaxed in order
  - Thus, by the path-relaxation lemma, the algorithm is correct
- **Analysis**
  - Can You Do It!?!!?
Dijkstra’s Algorithm

- Works only if the graph has no negative-weight edges
- This is essentially a weighted-version of BFS
  - Instead of a FIFO Queue (like you used for BFS in the lab), use a priority queue
  - Keys (in PQ) are the shortest-path weight estimates (d[v])
- In Dijkstra’s Algorithm, we have two sets of vertices
  - S: Vertices whose final shortest path weights are determined
  - Q: Priority queue: V \ S

Note: Looks a lot like Prim’s algorithm, but computing d[v], and using the shortest path weights as keys
Dijkstra’s Algorithm is greedy, since it always chooses the “lightest” vertex in V \ S to add to S

Dijkstra Analysis

- Analysis: Like Prim’s Algorithm, depends on the time it takes to perform priority queue operations.
- Suppose we use a binary heap.
  - extract-min: Called O(|V|) times
  - relax: Called O(|E|) times
- How long does each of these operations take?
- Dijkstra’s Algorithm Runs in O(E lg V), with a binary heap implementation.
- Better Heap implementations get it down to O(V lg V + E).
- Our “List/Container” implementation took O(V^2)

Dijkstra’s Algorithm

Dijkstra(V, E, w, s)
1 Init-Single-Source(V, s)
2 S ← ∅
3 Q ← V
4 while Q ≠ ∅
5 do u ← Extract-Min(Q)
6 S ← S ∪ {u}
7 for each v ∈ Adj[u]
8 do Relax(u, v, w)
Dijkstra Correctness.

**Loop Invariant**

- At the start of each iteration of the **while** loop
  \[\delta(s, v) = d[v] \forall v \in S\]
- **Initially**: \(S = \emptyset\), so this is trivially true
- **At end**: \(S = V\), so we have the shortest path weights
- **Maintenance**: Must show that \(d[u] = \delta(s, u)\) when \(u\) is added to \(S\)

**We’ll Give Proof (If Time)**