Shortest paths

- Generalization of BFS to handle weighted graphs
- Directed weighted graph $G = (V, E, w)$
- Weight function $w : E \rightarrow \mathbb{R}$
- Weight of path $p = v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_k$

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

- Shortest-path weight $u \leadsto v$

$$\delta(u, v) = \begin{cases} 
\min\{w(p) : u \leadsto v\} & \text{if there exists a path } p = u \leadsto v \\
\infty & \text{otherwise}
\end{cases}$$

- Shortest-path $u \leadsto v$

any path $p$ such that $w(p) = \delta(u, v)$
Shortest paths

- **Single-source shortest path problem (SSSP):**
  
  \[ \text{find shortest-paths from a given source vertex } s \in V \text{ to every vertex } v \in V \]  

- Most basic SSSP algorithm: **Bellman-Ford algorithm** *(discussed next)*

- **Variants:**
  - **Single-destination:** find shortest-paths to a given destination vertex 
    *(reverse the direction of each edge to become the single-source problem)*
  - **Single-pair:** find shortest-path from \( u \) to \( v \) 
    *(no way know that’s better in worst case than solving single-source)*
  - **All-pairs:** find shortest-paths from \( u \) to \( v \) for all \( u, v \in V \). 
    *(By running Bellman-Ford once for each vertex, cost \( O(V^2 E) = O(V^4) \) on dense graph. Can do better, see Chapter 25 of CLRS, 3ed)*
Shortest paths

Well-definedness

- Negative-weight edges are OK, as long as no negative-weight cycles reachable from the source. Otherwise, can always get a shorter path by going around the cycle again.

- The shortest path problem is ill-posed in graph with negative-weight cycle

- Bellman-Ford algorithm can detect and report the existence of negative-weight cycle
Shortest paths

- Optimal substructure property of SSSP:
  
  *subpaths of shortest-paths are shortest-paths.*

  \[ \text{Proof.} \text{ If some subpath were not a shortest path, could substitute it and create a shorter total path.} \]

- Thus, will see greedy and dynamical programming algorithms.
Shortest paths

- **Notation:**
  - \( d[v] \): shortest-path estimate
  - \( \pi[v] \): predecessor of \( v \)

- **Output of SSSP algorithms**
  \[
  d[v] = \delta(s, v) = \text{shortest-path weight } s \leadsto v \\
  \pi[v] = \text{predecessor of } v \text{ on a shortest path from } s.
  \]
Shortest paths

Two key components of shortest-path algorithms:

▶ Initialization

for every vertex v in V
    d[v] = inf
    pi[v] = nil
endfor

d[s] = 0 // s = source vertex

▶ Relaxing an edge \((u, v)\)

Can we improve the shortest-path estimate \(d[v]\) by going through \(u\) and taking the edge \((u, v)\)?

if \(d[v] > d[u] + w(u,v)\)
    \(d[v] = d[u] + w(u,v)\)
    \(pi[v] = u\)
endif
Shortest paths

Basic properties:

1. **Triangular inequality**
   
   for all \((u, v) \in E\), \(\delta(u, v) \leq \delta(u, x) + \delta(x, v)\)

2. **Upper-bound property**
   
   Always have \(d[v] \geq \delta(s, v)\) for all \(v\).
   
   Once \(d[v] = \delta(s, v)\), it never changes

3. **No-path property**
   
   If \(\delta(s, v) = \infty\), then \(d[v] = \infty\) always

4. **Convergence property**
   
   If \(s \leadsto u \rightarrow v\) is a shortest-path, and \(d[u] = \delta(s, u)\). Then after “Relax \(u \rightarrow v\)”, \(d[v] = \delta(s, v)\)

5. **Path relaxation property**
   
   Let \(p = v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_k\) be a shortest-path. If we relax in order, \((v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)\), even intermixed with other relaxations, then \(d[v_k] = \delta(v_0, v_k)\)