SHORTEST PATHS

CS340

Shortest Paths

- **Input**: Graph with edge weights
- **Weight of path p** = sum of edge weights on path p
- Shortest-path weight = δ(u,v)

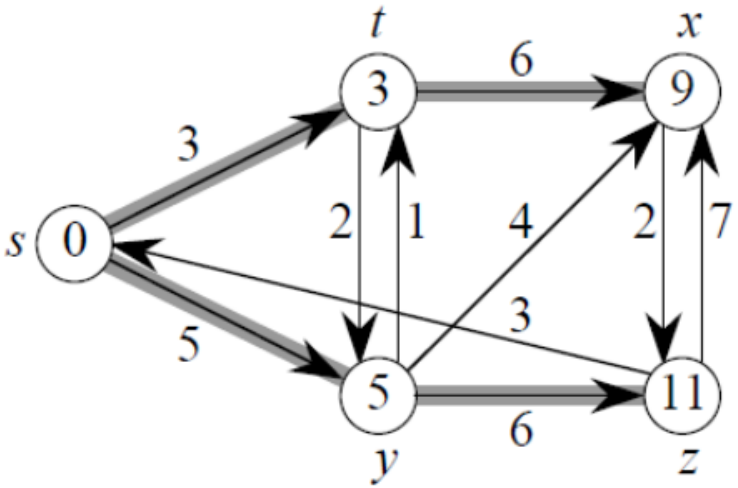
We define the ***shortest-path weight*** $\delta(u, v)$ from u to v by

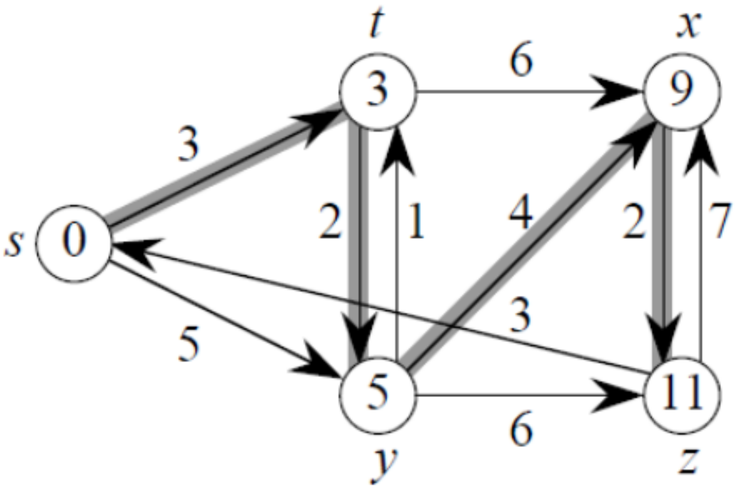
$$\delta(u, v) = \begin{cases} \min\{w(p) : u \overset{p}{\leadsto} v\} & \text{if there is a path from } u \text{ to } v, \\ \infty & \text{otherwise}. \end{cases}$$

A ***shortest path*** from vertex u to vertex v is then defined as any path p with weight $w(p) = \delta(u, v)$.

Shortest paths from s

- Shortest paths might not be unique
- Shortest paths form a tree





Variants

- Single-source shortest-paths problem
  - Given a graph G = (V,E) we want to find a shortest path from a given source vertex to all other vertices
- Single-pair shortest-path problem
  - All known algorithms for this problem have the same worst-case asymptotic running time as the best single-source algorithms.
- All-pairs shortest-paths problem
  - Find a shortest path from u to v for every pair of vertices u and v.

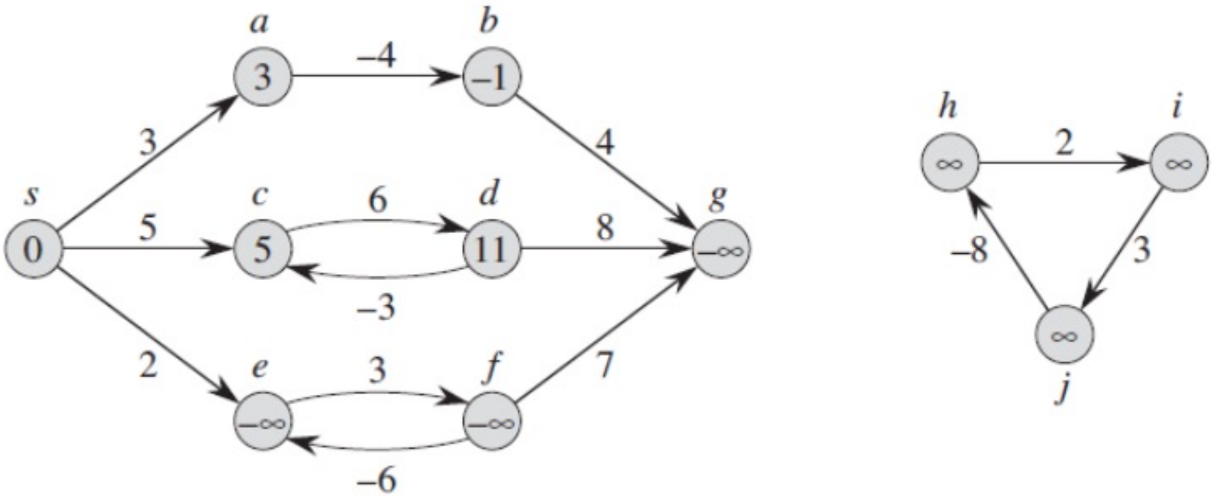
Optimal substructure of a shortest path

- A shortest path between two vertices contains other shortest paths within it.

Negative-weight edges

- OK, as long as no negative-weight cycles are reachable from the source.
- If we have a negative-weight cycle, we can just keep going around it, and get $\delta(s,v) = -\infty$ for all v on the cycle.
- Some algorithms work only if there are no negative-weight edges in the graph.
  - We'll be clear when they're allowed and not allowed.

Negative-Weight Cycles



- (e,f) is a negative-weight cycle reachable from s.
- Notice that the distance from s to g is -infinity.

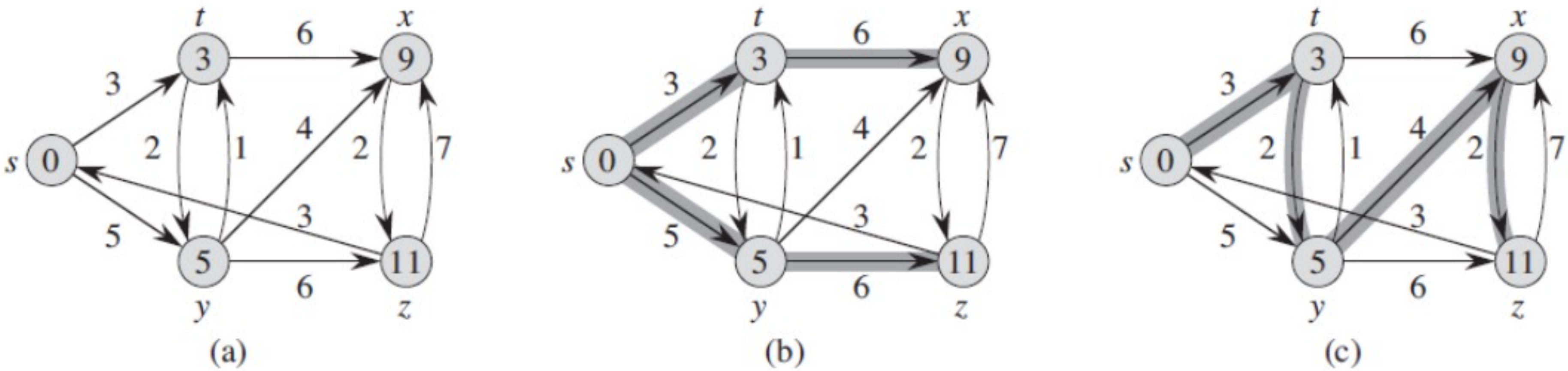
Cycles

- Shortest paths can’t contain cycles.
- Negative-weight
  - Already ruled out negative-weight cycles.
- Positive-weight
  - We can get a shorter path by omitting the cycle.
- Zero-weight
  - No reason to use them. Assume that our solutions won’t use them.

Output of single-source shortest-path algorithm

- For each vertex $v \in V$:
- $v.d = \delta(s, v)$
  - Initially $v.d = \infty$.
  - Reduces as algorithms progress. But always maintain $v.d \geq$ shortestDistance(s,v)
  - While running algorithm, $v.d$ is a shortest-path estimate.
- $v.\pi$ predecessor of v on a shortest path from s.
  - If no predecessor, $v.\pi$ = NIL.
  - π induces a shortest-path tree.

Output of single-source shortest-path algorithm



Shortest paths, and shortest path trees are not necessarily unique.

Initialization

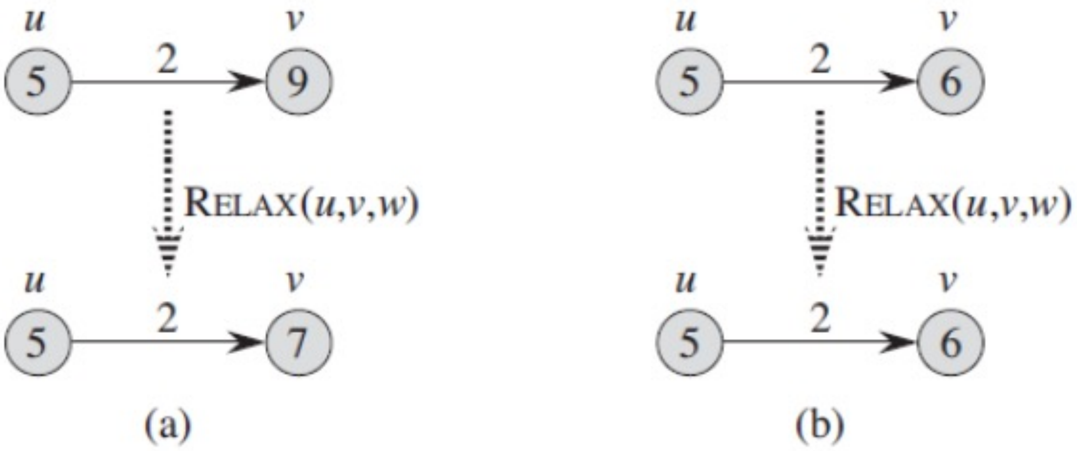
- All the shortest-paths algorithms start with INIT-SINGLE-SOURCE.

INITIALIZE-SINGLE-SOURCE(G, s)

for each vertex $v \in G.V$
    $v.d = \infty$
    $v.\pi = \text{NIL}$
$s.d = 0$

Relaxing an edge

- Can we improve the shortest-path estimate for v by going through u and taking (u,v)?



Relaxing an edge

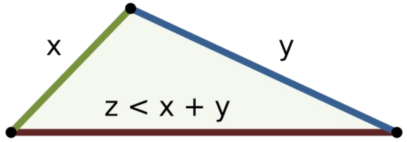
RELAX(u, v, w)

if $v.d > u.d + w(u, v)$

$\quad v.d = u.d + w(u, v)$

$\quad v.\pi = u$

Properties of shortest paths and relaxation



- Triangle inequality
- Upper-bound property
  - We always have $v.d \geq \delta(s,v)$ for all vertices v, and once v.d achieves the value $\delta(s,v)$ it never changes.
- No-path property
  - If there is no path from s to v, then we always have $v.d=\infty$.
- Convergence property
  - If $s \rightsquigarrow u \rightarrow v$ is a shortest path in G for some $u,v \in V$, and if $u.d=\delta(s,u)$ at any time prior to relaxing edge (u,v), then $v.d=\delta(s,v)$ at all times afterward.

Properties of shortest paths and relaxation

- Path-relaxation property
  - If $p = \langle v_0, v_1, \ldots, v_k \rangle$ is a shortest path from $s = v_0$ to v_k, and we relax the edges of p in the order $(v_0, v_1), (v_1, v_2), \ldots (v_{k-1}, v_k)$. then $v_k.d = \delta(s, v_k)$. This property holds **regardless** of any other relaxation steps that occur, even if they are intermixed with relaxations of the edges of p.

Dijkstra's 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 weights (v.d).
- Have two sets of vertices:
  - S = vertices whose final shortest-path weights are determined
  - Q = priority queue = V-S.

Dijkstra's Algorithm

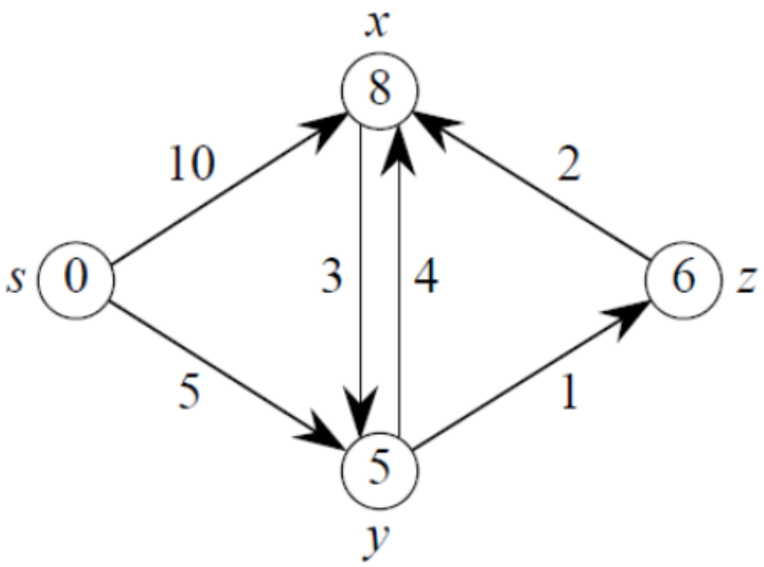
```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = Ø
Q = G.V
while Q ≠ Ø
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

Dijkstra's Algorithm

- Looks a lot like Prim’s algorithm, but computing v.d, and using shortest-path weights as keys.
- Dijkstra’s algorithm can be viewed as greedy, since it always chooses the “lightest” (“closest”) vertex in V-S to add to S.

Dijkstra's Algorithm

Step	s	x	y	Z
init	s.d=0 s.π=nil	x.d=∞ x.π=nil	y.d=∞ y.π=nil	z.d=∞ z.π=nil
1	s.d=0 s.π=nil	x.d=10 x.π=s	y.d=5 y.π=s	z.d=∞ z.π=nil
2	s.d=0 s.π=nil	x.d=9 x.π=y	y.d=5 y.π=s	z.d=6 z.π=y
3	s.d=0 s.π=nil	x.d=9 x.π=y	y.d=5 y.π=s	z.d=6 z.π=y
4	s.d=0 s.π=nil	x.d=8 x.π=z	y.d=5 y.π=s	z.d=6 z.π=y



Time Complexity of Dijkstra

- Time complexity depends on how it is implemented
- Matrix:
  - Each EXTRACT-MIN takes O(V) time to look through the array
  - There are V EXTRACT-MIN instructions for $O(V^2)$
- Priority Queue
  - The algorithm is only 1 line different from Prim
  - O(E lg V)