Single-Source Shortest Path

Chapter 24

Shortest Paths

- Finding the shortest path between two nodes comes up in many applications
  - Transportation problems
  - Motion planning
  - Communication problems
  - Six degrees of separation!

Shortest Paths

- In an unweighted graph, the cost of a path is just the number of edges on the shortest paths
- What algorithm have we already covered that can do this?
- In a weighted graph, the weight of a path between two vertices is the sum of the weights of the edges on a path

Shortest Paths Problems

- Input: a directed graph G = (V, E) and a weight function $w: E \to R$
- The weight of a path $p = v_0, v_1, v_2, \ldots, v_k$ is

Example

Variants

- Single Source Shortest Paths
- Single Destination Shortest Paths
- Single Pair Shortest Path
- All Pairs Shortest Paths

Subpaths

- Subpaths of shortest paths are shortest paths
- Lemma: If $p = v_0, v_1, v_2, \ldots, v_j, \ldots v_k$ is a shortest path from v_0 to v_k, then $p' = v_0, v_1, v_2, \ldots, v_j$ is a shortest path from v_0 to v_j

Negative Weight Edges

- Fine, as long as no negative-weight cycles are reachable from the source

Cycles

- Shortest paths can't contain cycles:
  - Already ruled out negative-weight cycles
  - Positive-weight → we can get a shorter weight by omitting the cycle
  - Zero-weight: no reason to use them → assume that our solutions will not use them

Output

- For each vertex v in V:
  - $d[v] = \delta (s,v)$
  - $\pi[v]$ = predecessor of v on a shortest path from s

Initialization

- All the shortest-paths algorithms start with

INIT-SINGLE-SOURCE(G, s)

for each $v \in G.V$

$\quad v.d = \infty$

$\quad v.\pi = \text{NIL}$

$s.d = 0$

Relaxation

- The process of relaxing an edge (u,v) consists of testing whether we can improve the shortest path to v found so far by going through u and, if so, updating d[v] and π [v]

RELAX(u, v, w)

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

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

$\quad v.\pi = u$

Single-Source Shortest-Paths

- For all single-source shortest-paths algorithms we'll look at:
  - Start by calling INIT-SINGLE-SOURCE
  - Then relax edges
- The algorithms differ in the order and how many times they relax each edge

Bellman-Ford Algorithm

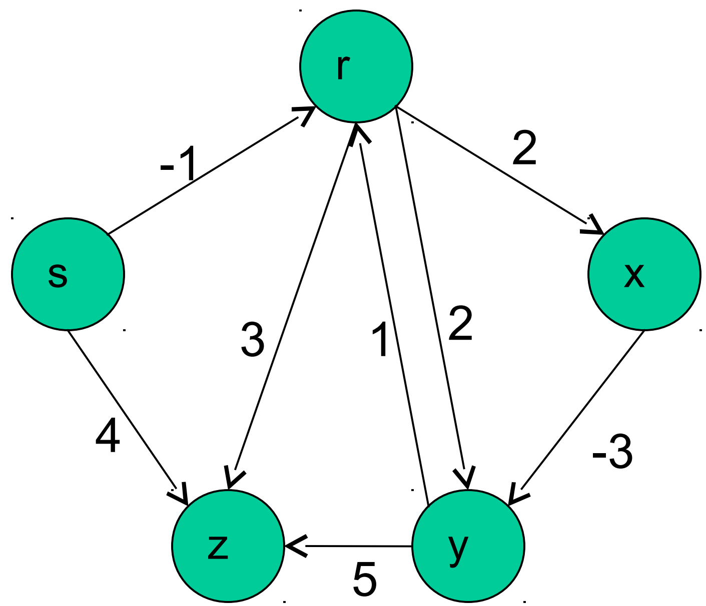
- Allows negative-weight edges
- Computes d[v] and π [v] for all v in V
- Returns true if no negative-weight cycles are reachable from s, false otherwise

BELLMAN FORD

```
BELLMAN-FORD(G, w, s)
INIT-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| - 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

- Time:

Example



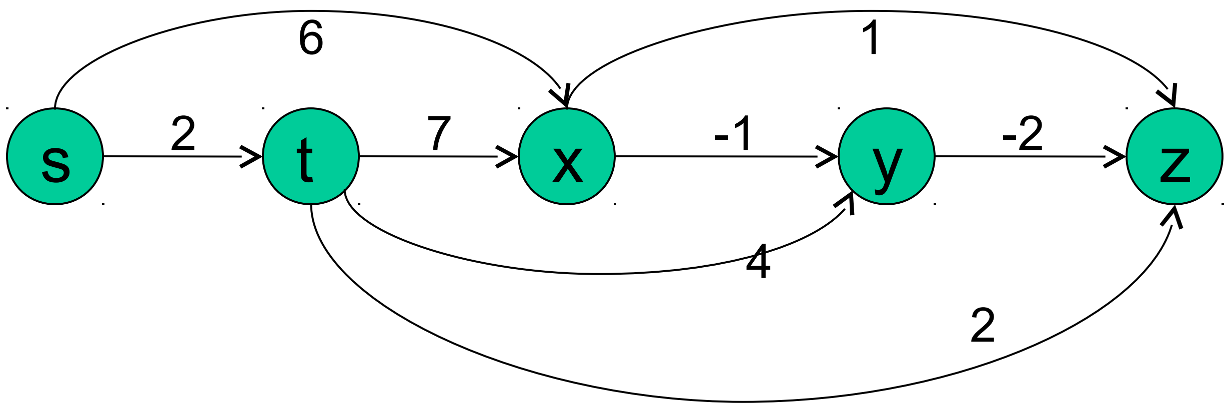
Single-Source Shortest-Paths

- In a DAG!

DAG-SHORTEST-PATHS(G, w, s)

```
topologically sort the vertices
INIT-SINGLE-SOURCE(G, s)
for each vertex u, taken in topologically sorted order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

Example



Dijkstra's Algorithm

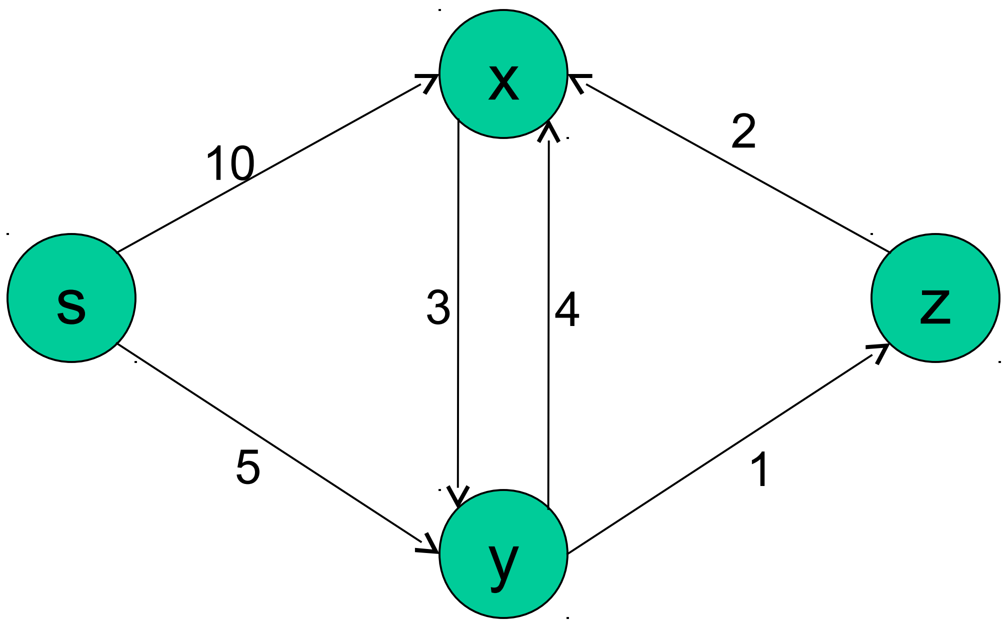
- No negative-weight edges
- Essentially a weighted version of BFS
  - Instead of a FIFO Queue, use a priority queue
  - Keys are shortest-path weights (d[v])
- Have two sets of vertices
  - S = vertices whose final shortest-path weights are determined
  - Q = priority queue = V - S

DIJKSTRA

$\text{DIJKSTRA}(G, w, s)$

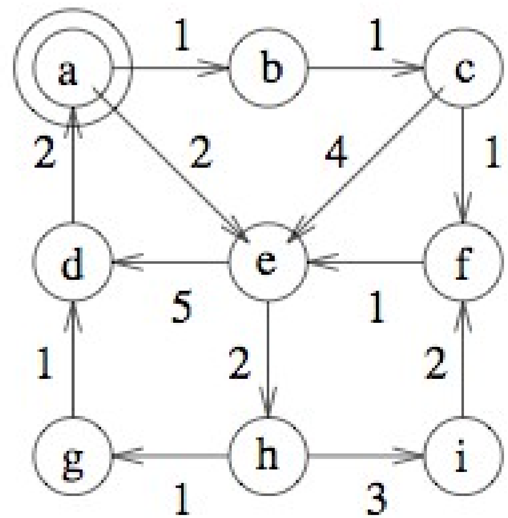
$\text{INIT-SINGLE-SOURCE}(G, s)$
$S = \emptyset$
$Q = G.V$ // i.e., insert all vertices into Q
while $Q \neq \emptyset$
    $u = \text{EXTRACT-MIN}(Q)$
    $S = S \cup \{u\}$
    for each vertex $v \in G.Adj[u]$
        $\text{RELAX}(u, v, w)$

Example



Your Turn

- What is the single-source shortest-path tree starting at a?



Question

- We are running one of these three algorithms on the graph below, where the algorithm has already processed the bold-face edges.
  - Prim's for the minimum spanning tree
  - Kruskal's for the minimum spanning tree
  - Dijkstra's shortest paths from *s*

Continued

- Which edge would be added next in Prim's algorithm
- Which edge would be added next in Kruskal's algorithm
- Which vertex would be marked next in Dijkstra's algorithm?