# Graphs: Shortest Paths

Atul Gupta

# Problems Specific with Graphs

- Minimum Spanning Tree
- Path Problems
  - Simple Paths
  - Shortest Path Problem
    - Single source shortest paths
    - All-pair shortest paths
  - Find Cycles
  - Euler Path and Circuit Problem
  - Hamiltonian Path and Circuit Problem (or TSP)
- Graph Coloring
- Connected Components
- Isomorphic graphs
- Search Graphs

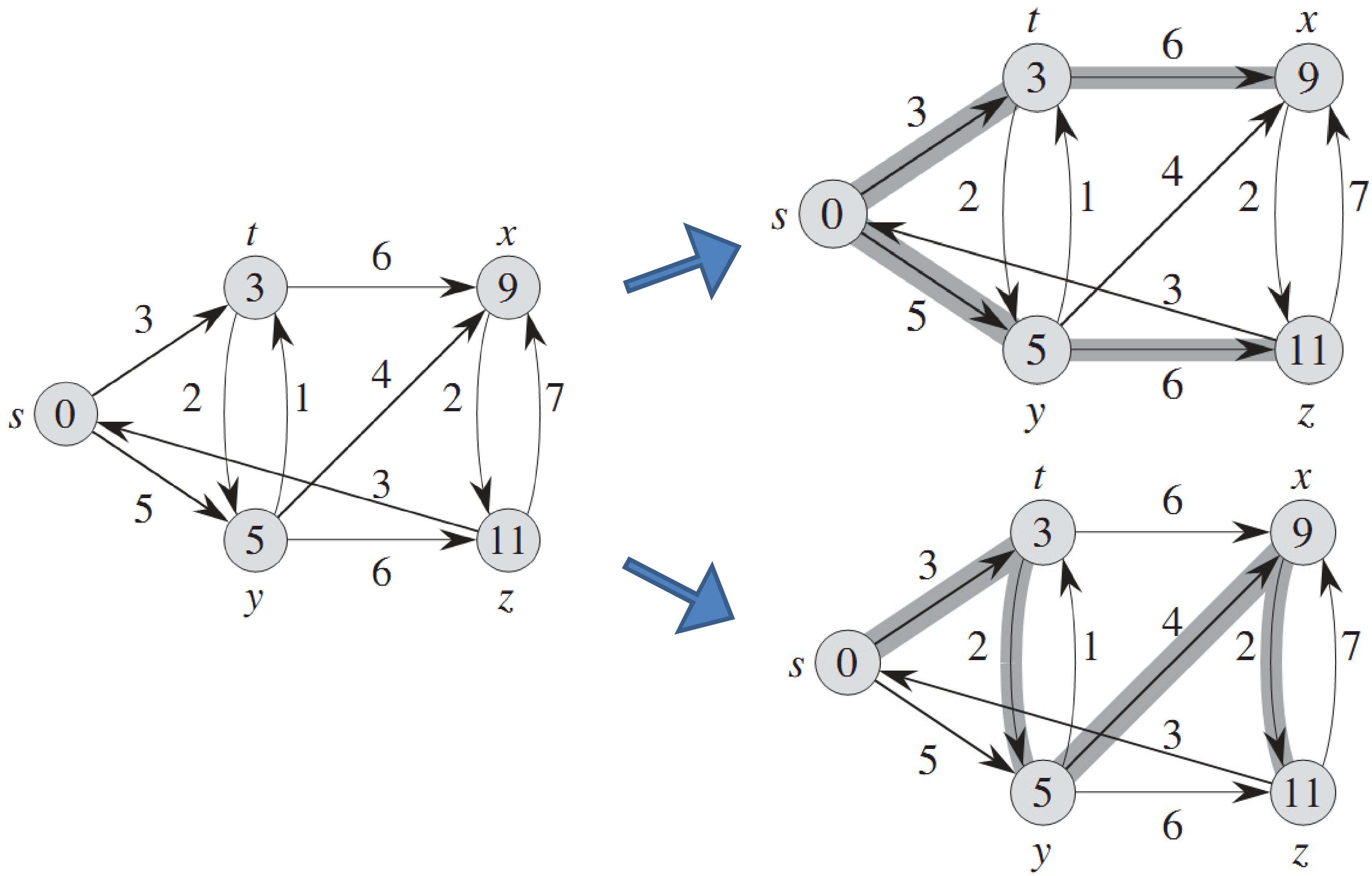
# Shortest Path

- Given a weighted graph G(V, E) with weight function w: E → R
- Weight *w(p)* of path p $(v_0, v_1, v_2, \ldots, v_k)$ is given by

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

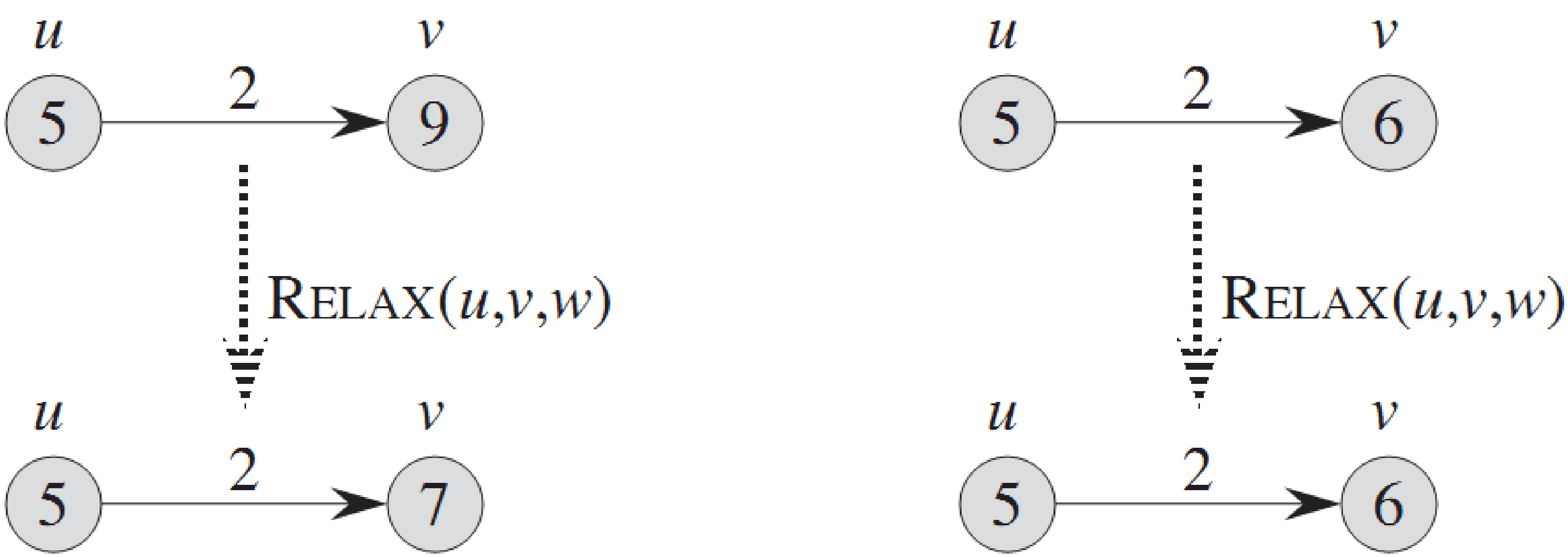
- A shortest path between two nodes will be a path with minimum w(p)

# Single Source Shortest Paths



# Shortest Paths

- Main Idea: Relaxing edges in the graph
  - Two cases




(a) (b)

# Relaxing an Edge (u, v) in the Graph

RELAX$(u, v, w)$

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

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

$\quad v.\pi = u$

# Main Components of Shortest Path Algorithms

INITIALIZE-SINGLE-SOURCE$(G, s)$

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

RELAX$(u, v, w)$

**if** $v.d > u.d + w(u, v)$
    $v.d = u.d + w(u, v)$
    $v.\pi = u$

# Dijkestra’s Algorithm

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

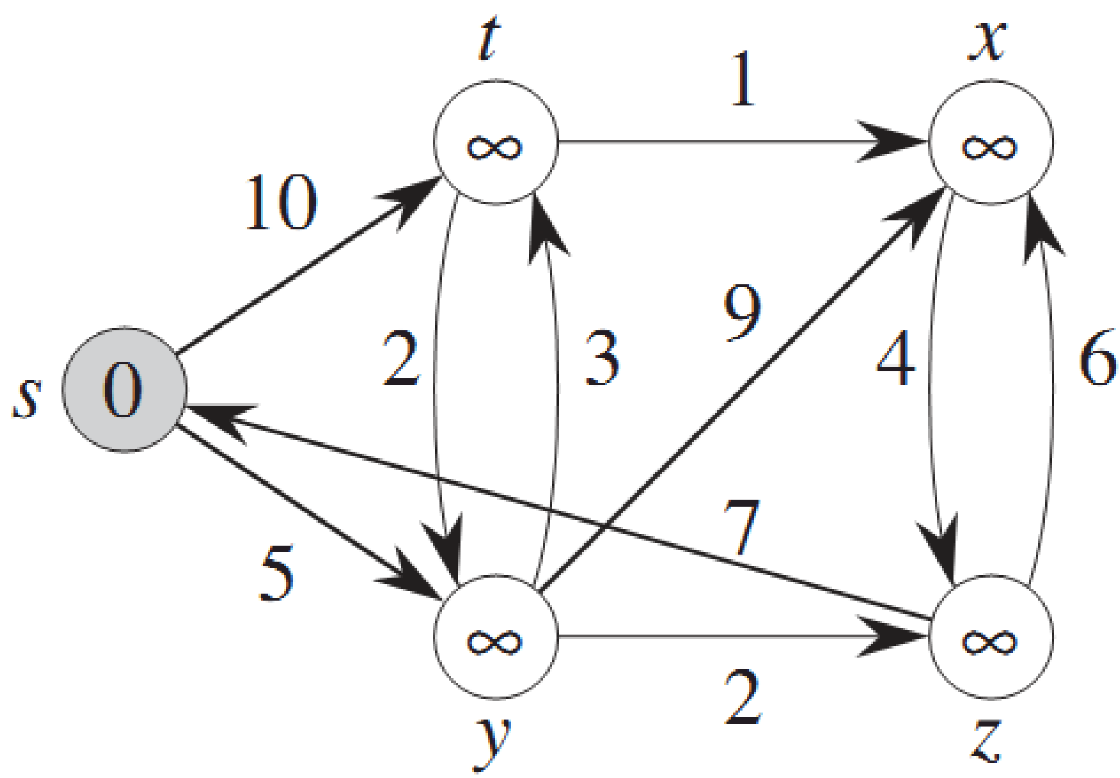
# Dijkestra's SP vs. Prim's MST

```
MST-PRIM(G, w, r)
for each u ∈ G.V do
    key[u] ← ∞
    π[u] ← NIL
key[r] ← 0
Q = G.V
while Q ≠ Ø do
        u ← EXTRACT-MIN(Q)
        for each v ∈ Adj[u] do
            if v ∈ Q and w(u, v) < key[v]
                then π[v] ← u
                key[v] ← w(u, v)
```

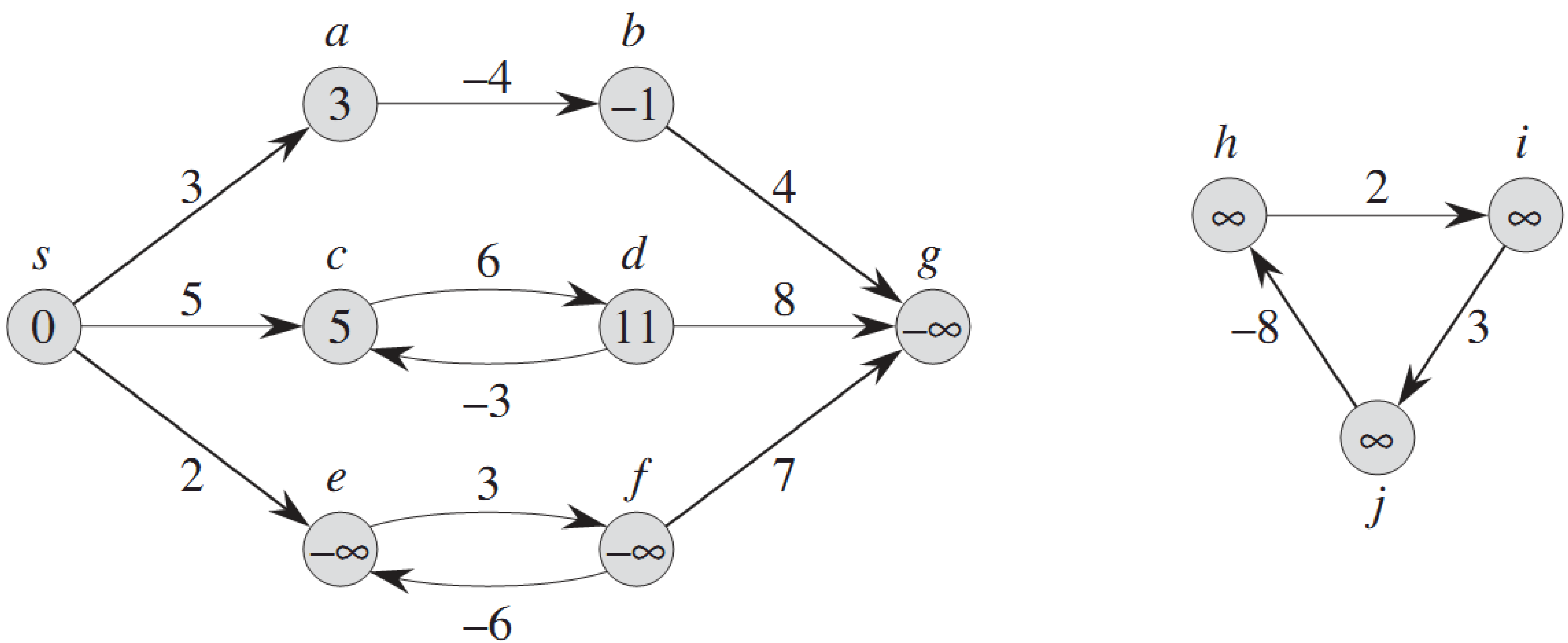
```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

```
RELAX(u, v, w)
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.π = u
```

# An Example: Dijkestra's Algo



# Negative Weight Edges



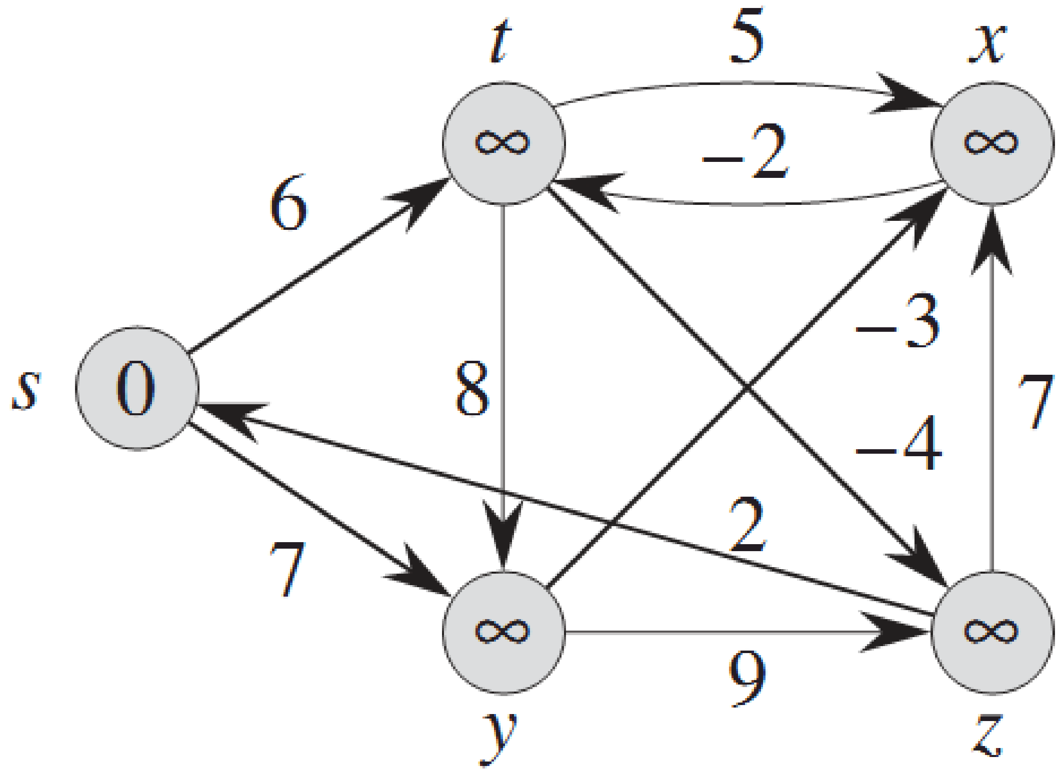
# Negative Weight Edges and Cycles

- If a graph contains a "negative cycle", i.e., a cycle whose edges sum to a negative value, then walks of arbitrarily low weight can be constructed, i.e., there may be no *shortest* path
- If a graph contains no negative cycle but some edges with negative weights, Bellman-Ford algorithm will be useful
- Bellman-Ford algorithm can also detect negative cycles and report their existence

# Bellman-Ford Algorithm

```
BELLMAN-FORD(G, w, s)
INITIALIZE-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
```

# Example: Bellman-Ford Algo



# Summary

- Here we presented two algorithms for single source shortest paths
  - Dijkestra's
  - Bellman-Ford's
- Both make use of "Edge Relaxation" approach
- Dijkestra's algorithm is faster but can not be used with graphs having negative weight edges
- Bellman-Ford's algorithm can work with graphs having negative weight edges but not the "negative cycles"
- The Dijkestra algorithm is an example of greedy approach whereas Bellman-Ford algorithm in an example of dynamic programming approach.