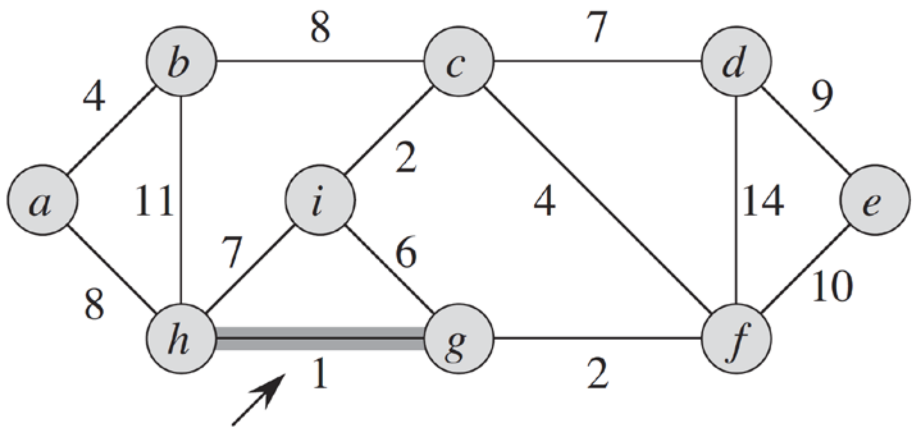
# CS 457, Fall 2019

Drexel University, Department of Computer Science

Lecture 17

# Minimum Spanning Tree of a Weighted Graph

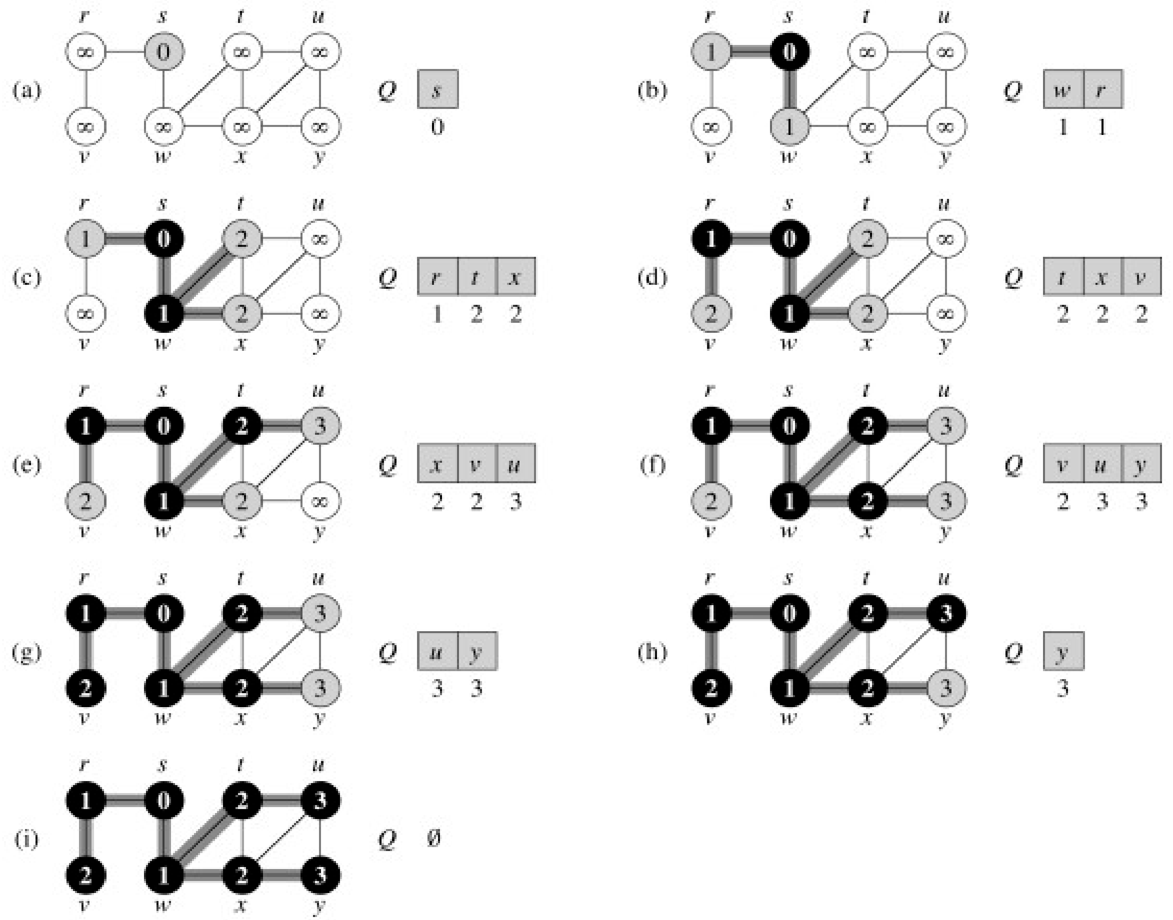
- Let *G*=(*V*,*E*) be a graph on $n$ vertices, $m$ edges, and a weight $w$ on edges in *E*.
- Sub-graph *T=(V,E')* with $E' \subseteq E$ with no cycles is a spanning tree
- The weight of *T* is the sum of the weights of its edges: $w(T) = \sum_{(u,v) \in E'} w(u,v)$



# BFS Algorithm

**BFS** $(G, s)$

```
for each u ∈ G.V − {s}
        u.color = WHITE
        u.d = ∞
        u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
        u = DEQUEUE(Q)
        for each v ∈ G.Adj[u]
                if v.color == WHITE
                        v.color = GRAY
                        v.d = u.d + 1
                        v.π = u
                        ENQUEUE(Q, v)
        u.color = BLACK
```



# DFS Algorithm

**DFS ($G$)**

**for** each vertex $u \in G.V$
$u$.color = WHITE
$u$.π = NIL
time = 0
**for** each vertex $u \in G.V$
**if** $u$.color = WHITE
DFS-Visit($G$,$u$)

**DFS-Visit($G$,$u$)**

time = time + 1
$u$.d = time
$u$.color = GRAY
**for** each vertex $v \in G.Adj[u]$
**if** $v$.color == WHITE
$v$.π = $u$
DFS-Visit($G$, $v$)
$u$.color = BLACK
time = time + 1
$u$.f = time

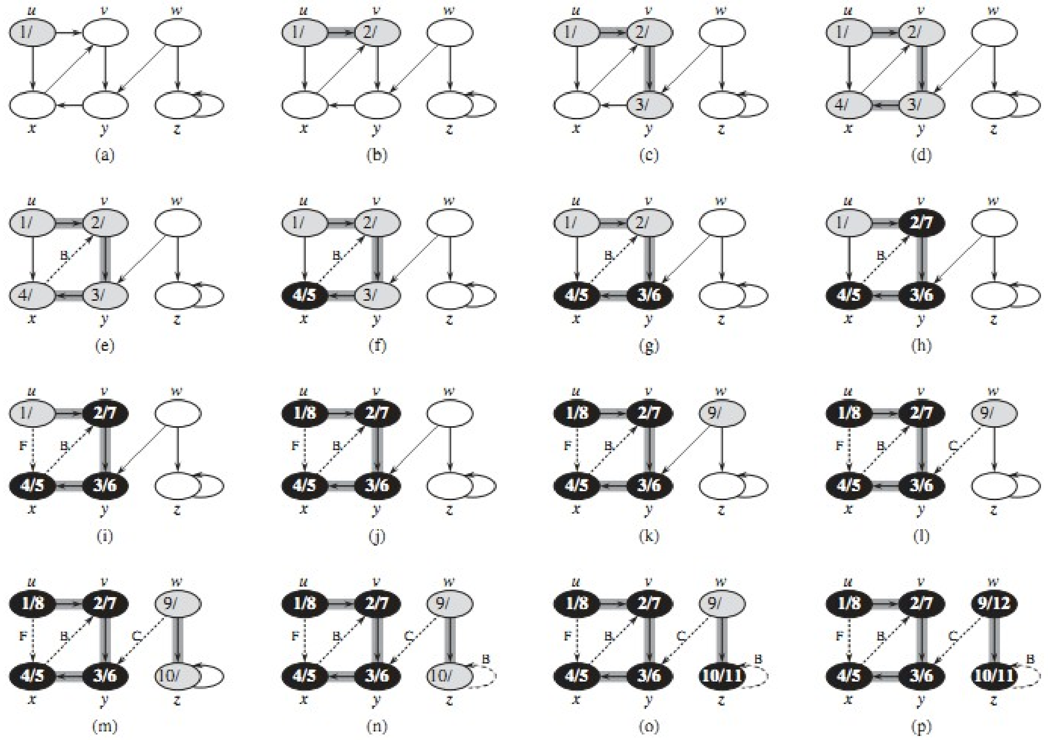
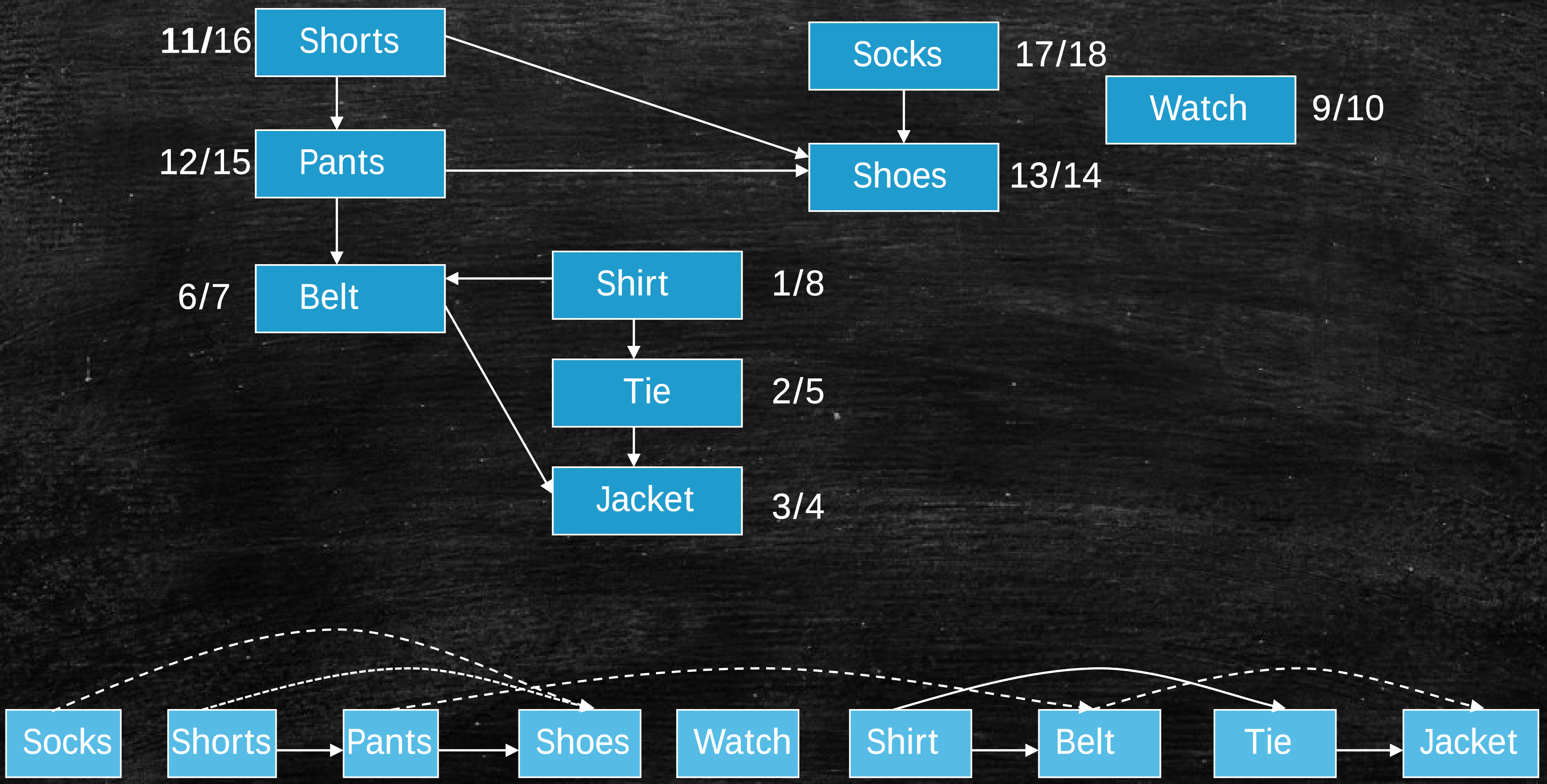


**Figure 22.4** The progress of the depth-first-search algorithm DFS on a directed graph. As edges are explored by the algorithm, they are shown as either shaded (if they are tree edges) or dashed (otherwise). Nontree edges are labeled B, C, or F according to whether they are back, cross, or forward edges. Timestamps within vertices indicate discovery time/finishing times.

# Example


11/16
Shorts
Socks
17/18
Watch
9/10
12/15
Pants
Shoes
13/14
6/7
Belt
Shirt
1/8
Tie
2/5
Jacket
3/4
Socks
Shorts
Pants
Shoes
Watch
Shirt
Belt
Tie
Jacket

# Topological Sort Algorithm

TopologicalSort($G$)

1. call DFS(G) to compute finishing times $v.f$ for each vertex $v$
2. as each vertex is finished, insert it onto the front of a linked list
3. return the linked list of vertices

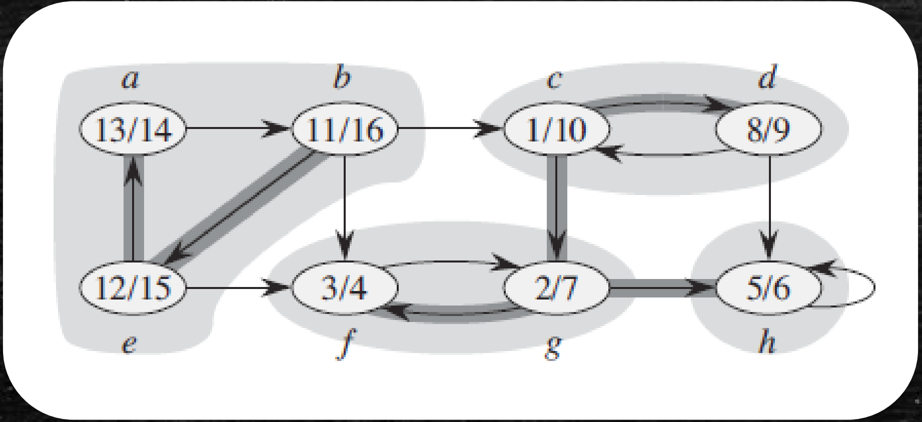
Running time

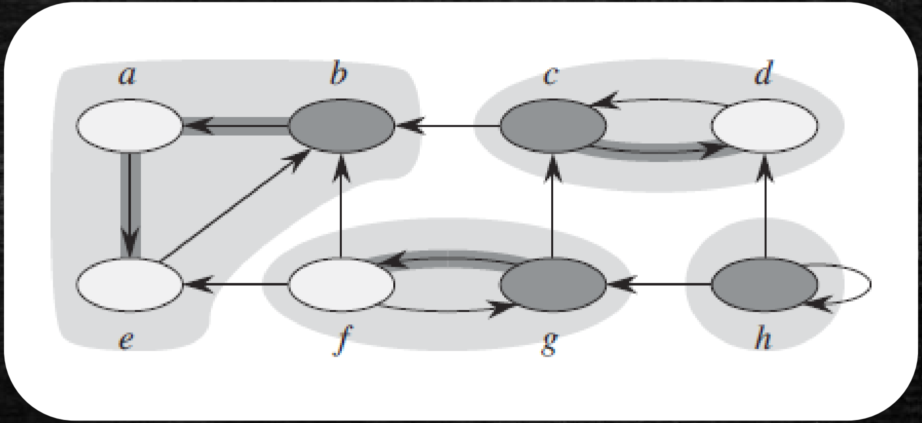
- DFS: $\Theta(n+m)$
- Insertion to linked list: $\Theta(n)$

# Strongly Connected Components

Strongly Connected Componenets($G$)

1. call DFS(G) to compute finishing times $v.f$ for each vertex $v$
2. compute $G^T$
3. call DFS($G^T$), but in the main loop of DFS, use decreasing order w.r.t. $v.f$
4. return the vertices of each tree in the depth-first forest formed in line 3 as a separate strongly connected component

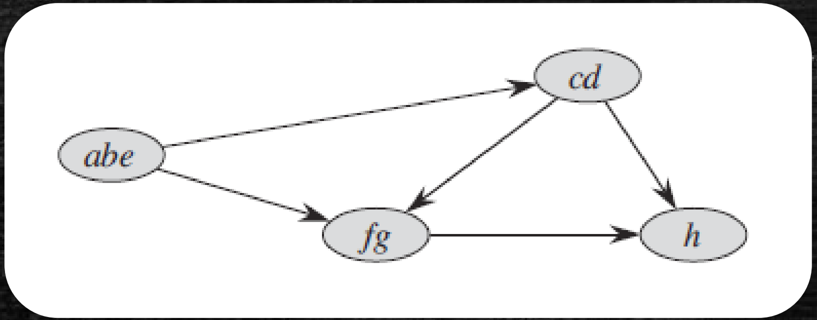


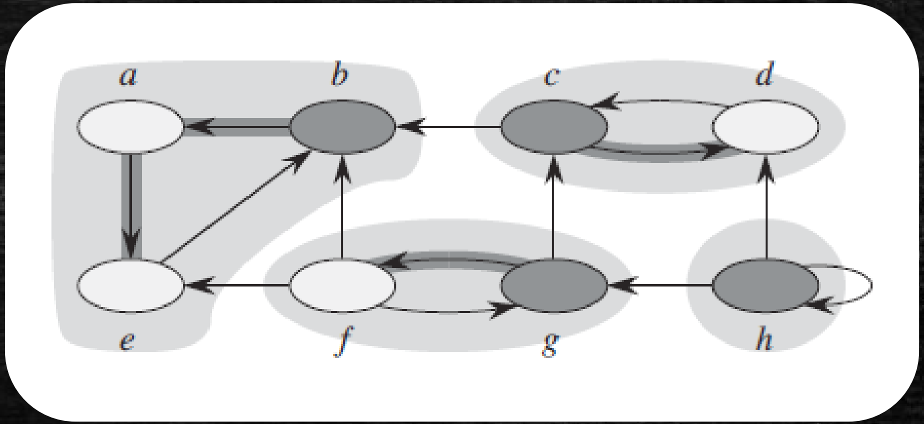



# Strongly Connected Components

Strongly Connected Componenets($G$)

1. call DFS(G) to compute finishing times $v.f$ for each vertex $v$
2. compute $G^T$
3. call DFS($G^T$), but in the main loop of DFS, use decreasing order w.r.t. $v.f$
4. return the vertices of each tree in the depth-first forest formed in line 3 as a separate strongly connected component



# Practice Problems

- A root of a DAG is a vertex r such that every other vertex of the DAG can be reached from r using a directed path. Give an algorithm that determines whether a given DAG has a root.

- Provide an algorithm that, given a directed acyclic graph $G = (V, E)$ and two vertices $s, t \in V$, returns the number of simple paths from $s$ to $t$ in $G$.

# Today’s Lecture

- Another example of a greedy algorithm that works well
- Single source shortest paths for weighted graphs

# Activity-Selection problem

- You are given a set of $n$ activities $S = \{a_1, a_2, \dots, a_n\}$, with each activity $a_i$ associated with a start time $s_i > 0$ and a finish time $f_i > s_i$
- Two activities $a_i$ and $a_j$ are compatible if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap, i.e., $s_i \geq f_j$ or $s_j \geq f_i$
- Assume that these activities are sorted so that $f_1 \leq f_2 \leq \cdots \leq f_n$
- Select a maximum-size subset of mutually compatible activities

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $s_i$ | 1 | 3 | 0 | 5 | 3 | 5 | 6 | 8 | 8 | 2 | 12 |
| $f_i$ | 4 | 5 | 6 | 7 | 9 | 9 | 10 | 11 | 12 | 14 | 16 |

- Dynamic programming solution:
  - Let $S_{ij}$ be the set of activities that start after activity $a_i$ ends and end before activity $a_j$ starts
  - Let $A_{ij}$ be a maximum set of mutually compatible activities in $S_{ij}$ and $c[i,j]$ be its size

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset\,, \\ \max\limits_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \emptyset\,. \end{cases}$$

# Activity-Selection problem

- You are given a set of $n$ activities $S = \{a_1, a_2, \dots, a_n\}$, with each activity $a_i$ associated with a start time $s_i > 0$ and a finish time $f_i > s_i$
- Two activities $a_i$ and $a_j$ are compatible if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap, i.e., $s_i \geq f_j$ or $s_j \geq f_i$
- Assume that these activities are sorted so that $f_1 \leq f_2 \leq \cdots \leq f_n$
- Select a maximum-size subset of mutually compatible activities

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $s_i$ | 1 | 3 | 0 | 5 | 3 | 5 | 6 | 8 | 8 | 2 | 12 |
| $f_i$ | 4 | 5 | 6 | 7 | 9 | 9 | 10 | 11 | 12 | 14 | 16 |

- Greedy solution:
  - Intuition: we should choose an activity that leaves the resource as available as possible
  - Choose the first activity to finish and repeat

# Single-Source Shortest Paths

- Given a weighted, directed graph $G = (V, E)$
- Weight function $w: E \to \mathbb{R}$ mapping edges to real-valued weights
- The weight $w(p)$ of a path $p = \langle v_0, v_1, \dots, v_k \rangle$ is $\sum_{i=1}^{k} w(v_{i-1}, v_i)$
- The shortest-path weight $\delta(u, v)$ from $u$ to $v$ is

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

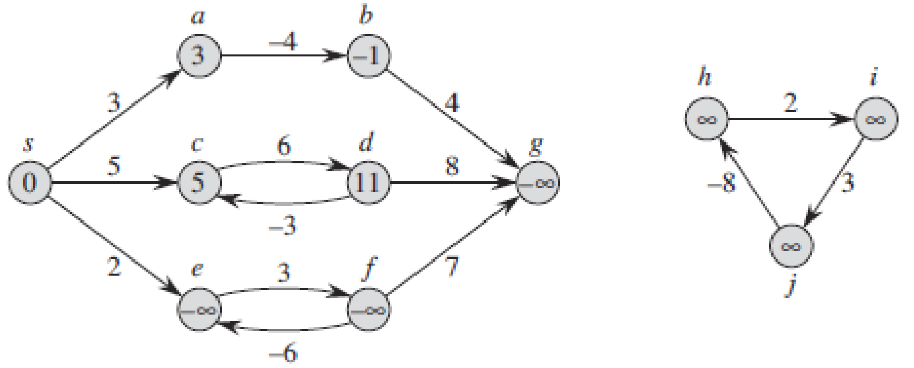
- A shortest-path from $u$ to $v$ is any path $p$ with weight $w(p) = \delta(u, v)$
- Problem: find a shortest path from a given source vertex $s \in V$ to each $v \in V$

# Single-Source Shortest Paths

- This problem exhibits optimal substructure, which is an indicator that the dynamic programming and the greedy methods may apply

***Lemma 24.1 (Subpaths of shortest paths are shortest paths)***
Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$, let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path from vertex $v_0$ to vertex $v_k$ and, for any $i$ and $j$ such that $0 \le i \le j \le k$, let $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ be the subpath of $p$ from vertex $v_i$ to vertex $v_j$. Then, $p_{ij}$ is a shortest path from $v_i$ to $v_j$.

- Negative-weight edges:
  - Do there exist negative-weight edges?
  - Do there exist negative-weight cycles?
  - Are these cycles reachable from the source?

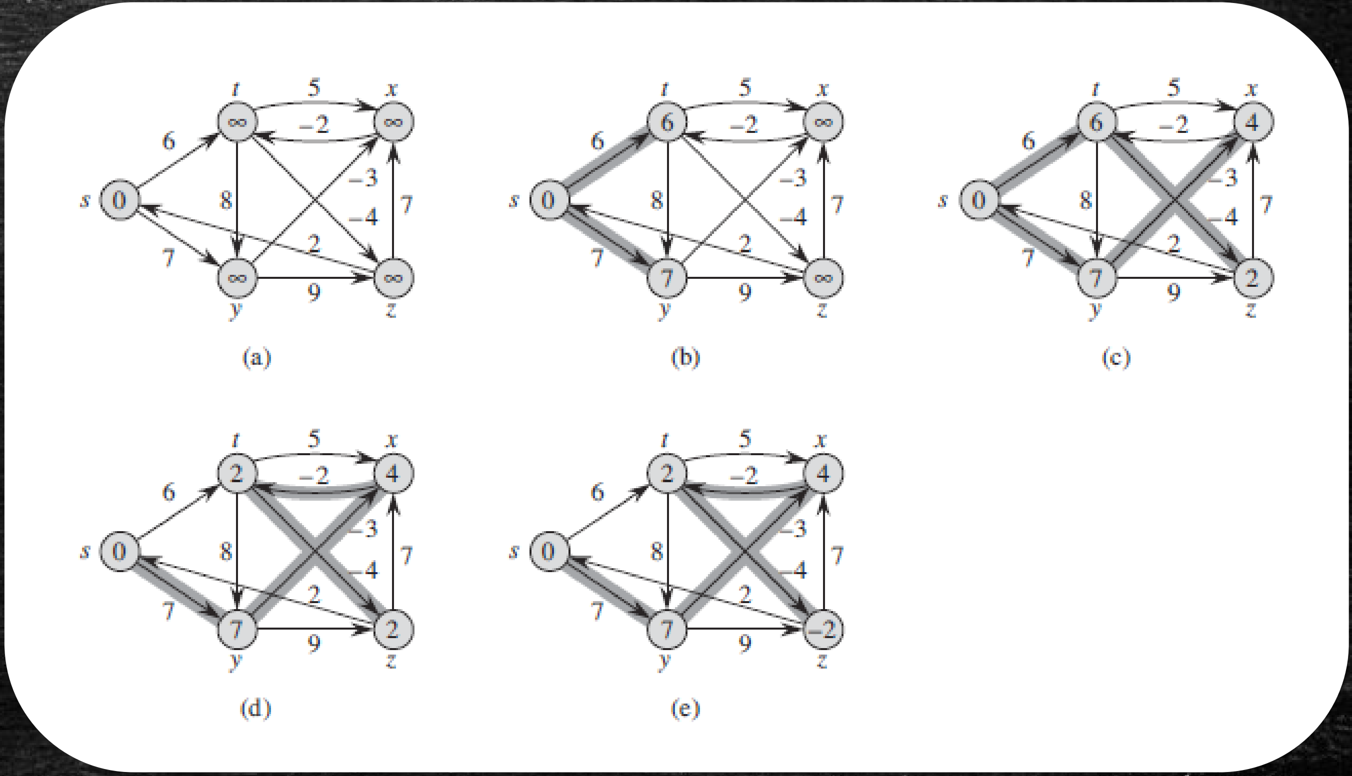


# Bellman-Ford Algorithm: time $O(nm)$

```
INITIALIZE-SINGLE-SOURCE(G, s)
  for each vertex v ∈ G.V
      v.d = ∞
      v.π = NIL
  s.d = 0
```

```
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
```

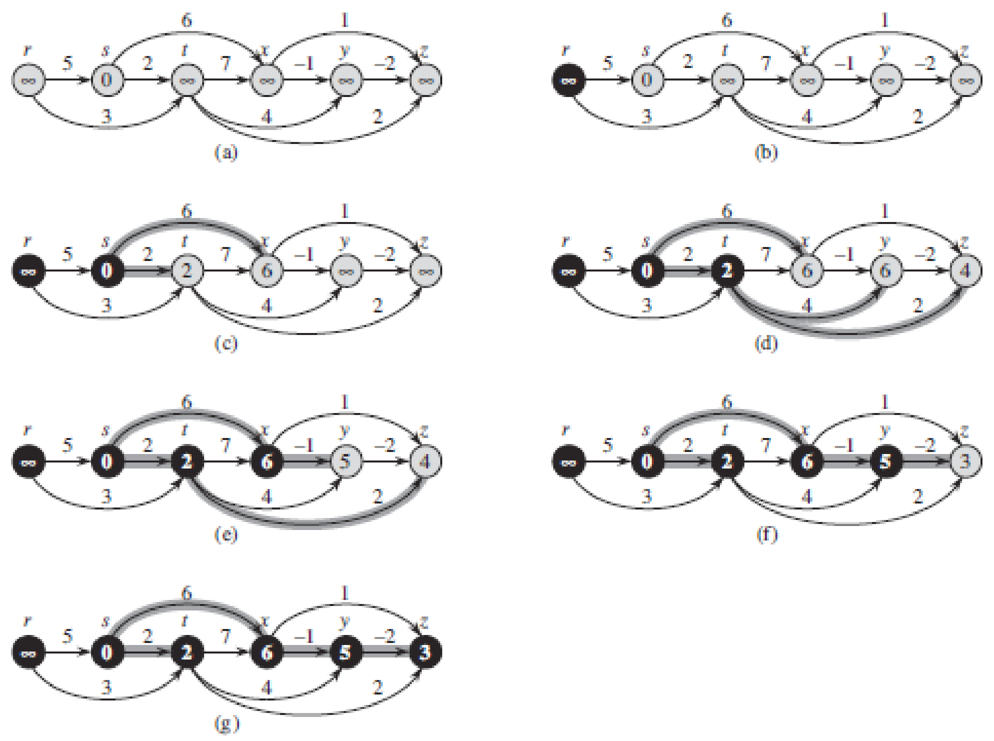
```
RELAX(u, v, w)
  if v.d > u.d + w(u, v)
      v.d = u.d + w(u, v)
      v.π = u
```



# SSSP in Directed Acyclic Graphs

- Solve the problem of single-source shortest paths for DAGs:

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

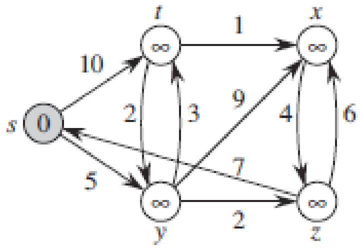



- Running time?
  - $O(n+m)$

# Dijkstra’s Algorithm

- Dijkstra’s algorithm assumes nonnegative weights!

```
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)
```



(a)

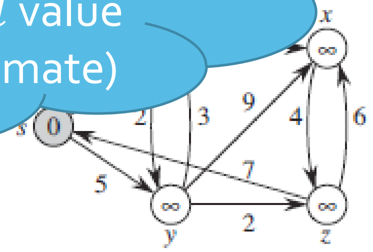
- Min-priority queue matters
- Time: $O(n^2 + m) = O(n^2)$
- This is a greedy algorithm

# Dijkstra's Algorithm

- Dijkstra's algorithm assumes nonnegative weights!

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

Extract vertex $v \in Q$ with minimum $v.d$ value (shortest path estimate)



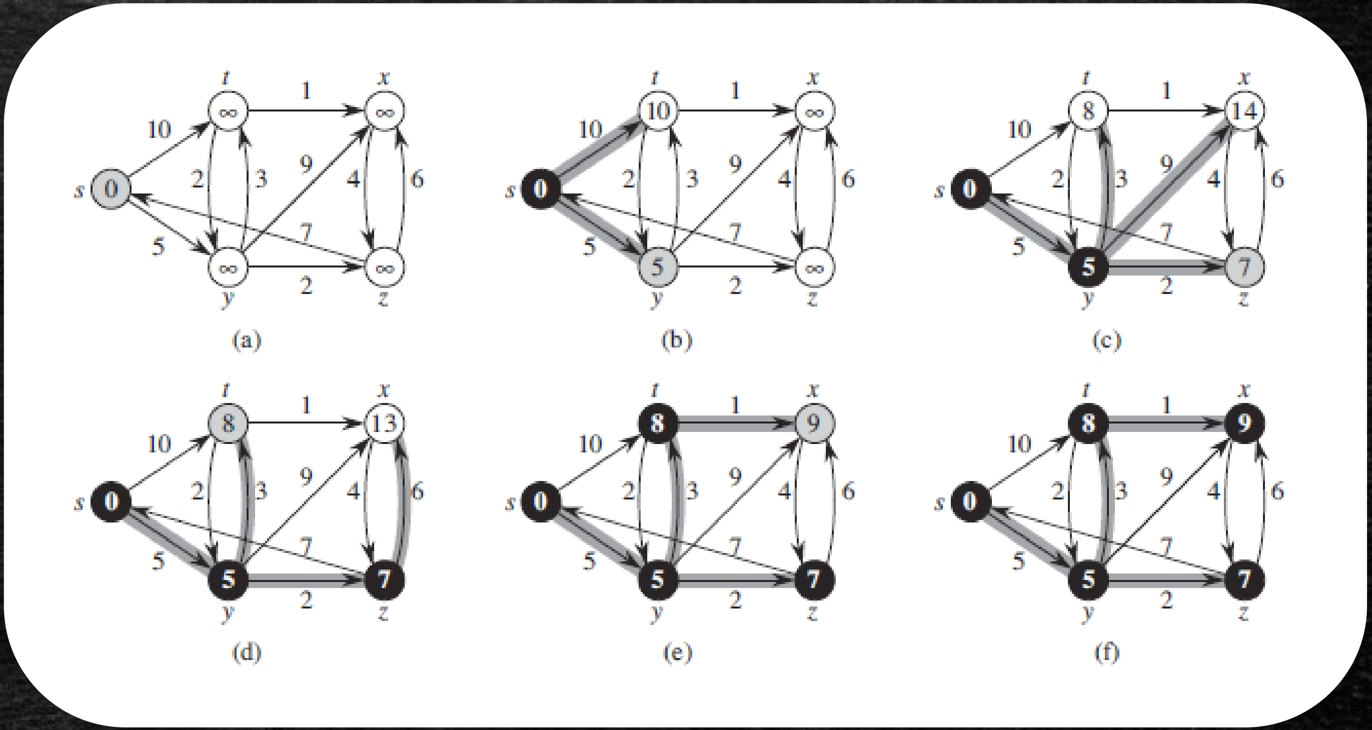

(a)

- Min-priority queue matters
- Time: $O(n^2 + m) = O(n^2)$
- This is a greedy algorithm

# Dijkstra’s Algorithm

- Dijkstra’s algorithm assumes nonnegative weights!

```
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)
```




- Min-priority queue matters
- Time: $O(n^2 + m) = O(n^2)$
- This is a greedy algorithm