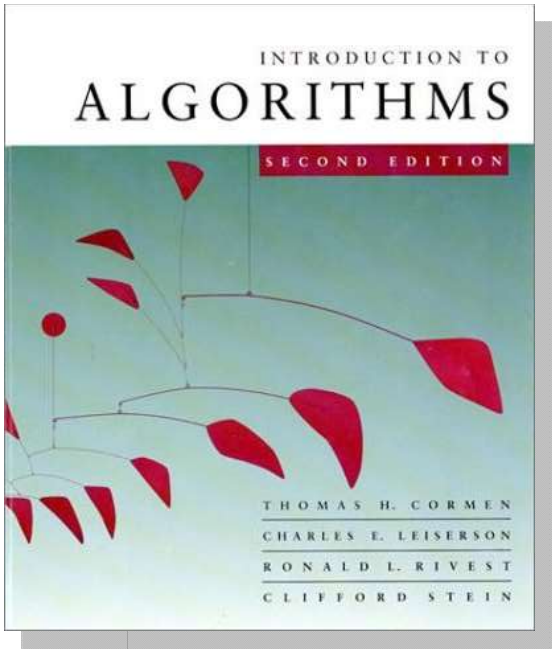
Introduction to Algorithms

6.046J/18.401J

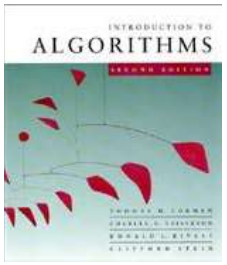


LECTURE 16

Greedy Algorithms (and Graphs)

- Graph representation
- Minimum spanning trees
- Optimal substructure
- Greedy choice
- Prim's greedy MST algorithm

Prof. Charles E. Leiserson



Graphs (review)

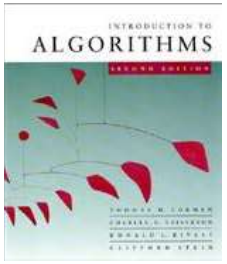
Definition. A ***directed graph (digraph)*** $G = (V, E)$ is an ordered pair consisting of

- a set V of ***vertices*** (singular: ***vertex***),
- a set $E \subseteq V \times V$ of ***edges***.

In an ***undirected graph*** $G = (V, E)$, the edge set E consists of *unordered* pairs of vertices.

In either case, we have $|E| = O(V^2)$. Moreover, if G is connected, then $|E| \geq |V| - 1$, which implies that $\lg|E| = \Theta(\lg V)$.

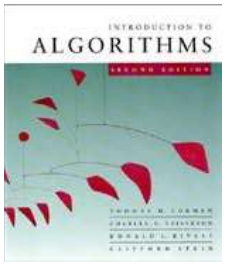
(Review CLRS, Appendix B.)



Adjacency-matrix representation

The ***adjacency matrix*** of a graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, is the matrix $A[1 \ldots n, 1 \ldots n]$ given by

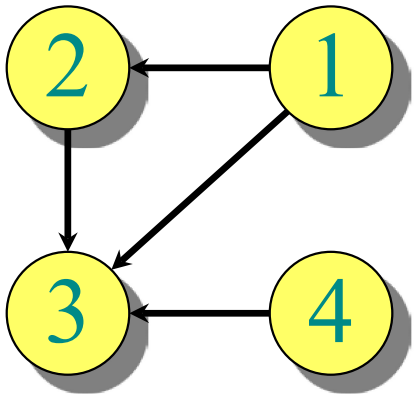
$$A[i, j] = \begin{cases} 1 & \text{if } (i, j) \in \text{E}, \\ 0 & \text{if } (i, j) \notin \text{E}. \end{cases}$$



Adjacency-matrix representation

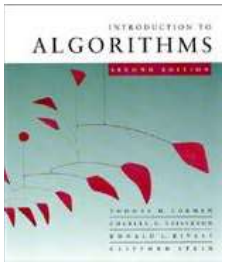
The ***adjacency matrix*** of a graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, is the matrix $A[1 \ldots n, 1 \ldots n]$ given by

$$A[i,j] = \begin{cases} 1 & \text{if } (i,j) \in \text{E}, \\ 0 & \text{if } (i,j) \notin \text{E}. \end{cases}$$



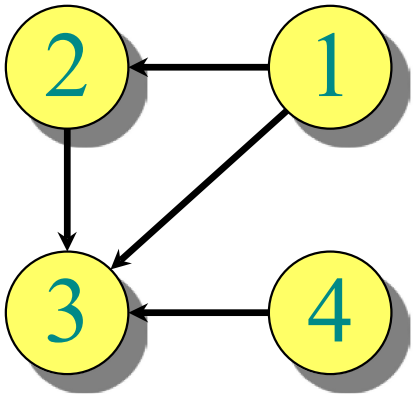
A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

$\Theta(V^2)$ storage $\Rightarrow$ ***dense*** representation.

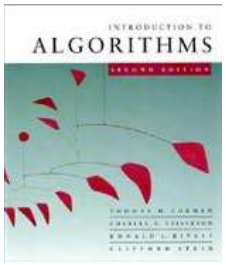


Adjacency-list representation

An ***adjacency list*** of a vertex $v \in V$ is the list $Adj[v]$ of vertices adjacent to v.

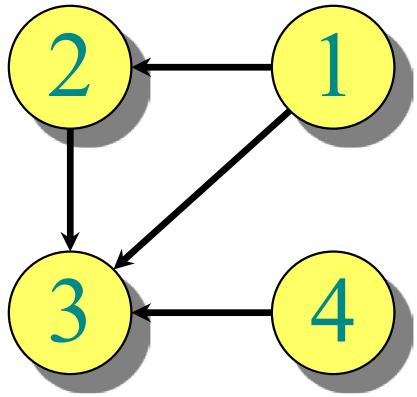


$Adj[1] = \{2, 3\}$
$Adj[2] = \{3\}$
$Adj[3] = \{\}$
$Adj[4] = \{3\}$



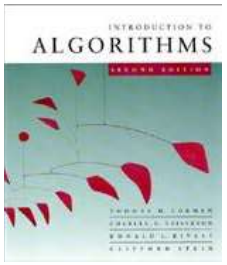
Adjacency-list representation

An ***adjacency list*** of a vertex $v \in V$ is the list $Adj[v]$ of vertices adjacent to v.



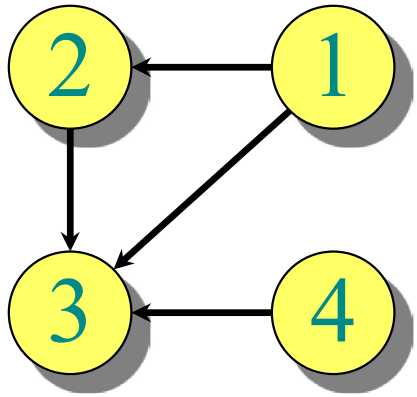
$Adj[1] = \{2, 3\}$
$Adj[2] = \{3\}$
$Adj[3] = \{\}$
$Adj[4] = \{3\}$

For undirected graphs, $|Adj[v]| = degree(v)$.
For digraphs, $|Adj[v]| = out\text{-}degree(v)$.



Adjacency-list representation

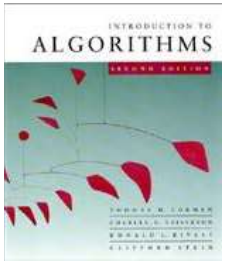
An ***adjacency list*** of a vertex $v \in V$ is the list $Adj[v]$ of vertices adjacent to v.



$Adj[1] = \{2, 3\}$
$Adj[2] = \{3\}$
$Adj[3] = \{\}$
$Adj[4] = \{3\}$

For undirected graphs, $|Adj[v]| = degree(v)$.
For digraphs, $|Adj[v]| = out\text{-}degree(v)$.

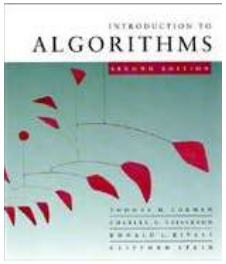
Handshaking Lemma: $\sum_{v \in V} degree(v) = 2|E|$ for undirected graphs $\Rightarrow$ adjacency lists use $\Theta(V + E)$ storage — a ***sparse*** representation.



Minimum spanning trees

Input: A connected, undirected graph $G = (V, E)$ with weight function $w : E \to \mathsf{R}$.

- For simplicity, assume that all edge weights are distinct. (CLRS covers the general case.)



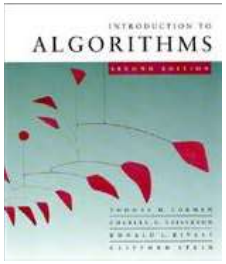
Minimum spanning trees

Input: A connected, undirected graph $G = (V, E)$ with weight function $w : E \to \mathsf{R}$.

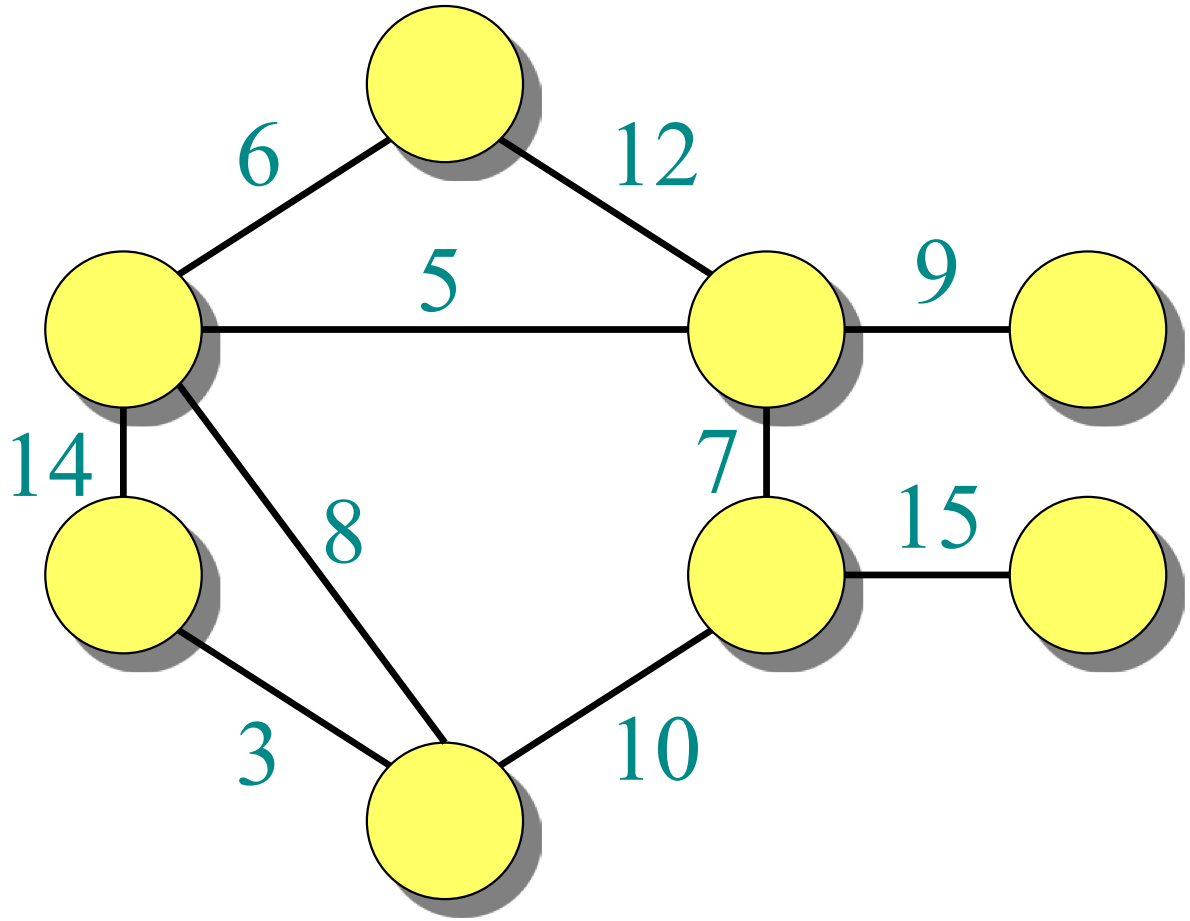
- For simplicity, assume that all edge weights are distinct. (CLRS covers the general case.)

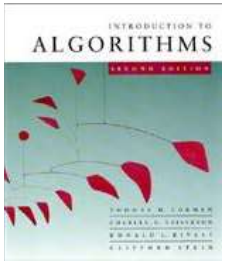
Output: A ***spanning tree*** T — a tree that connects all vertices — of minimum weight:

$$w(T) = \sum_{(u,v)\in T} w(u,v) .$$

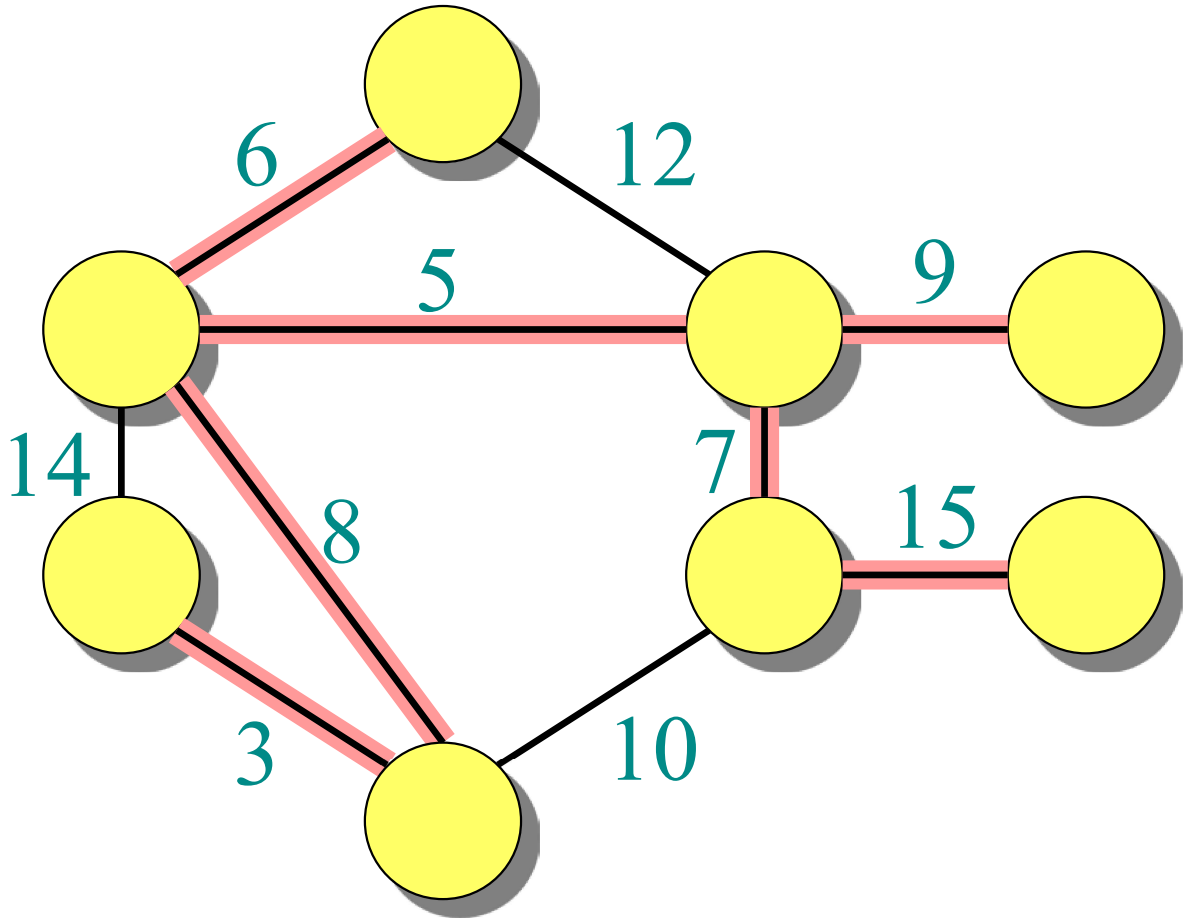


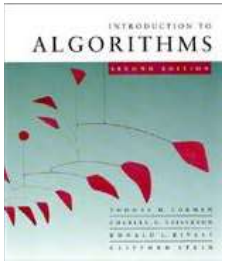
Example of MST





Example of MST

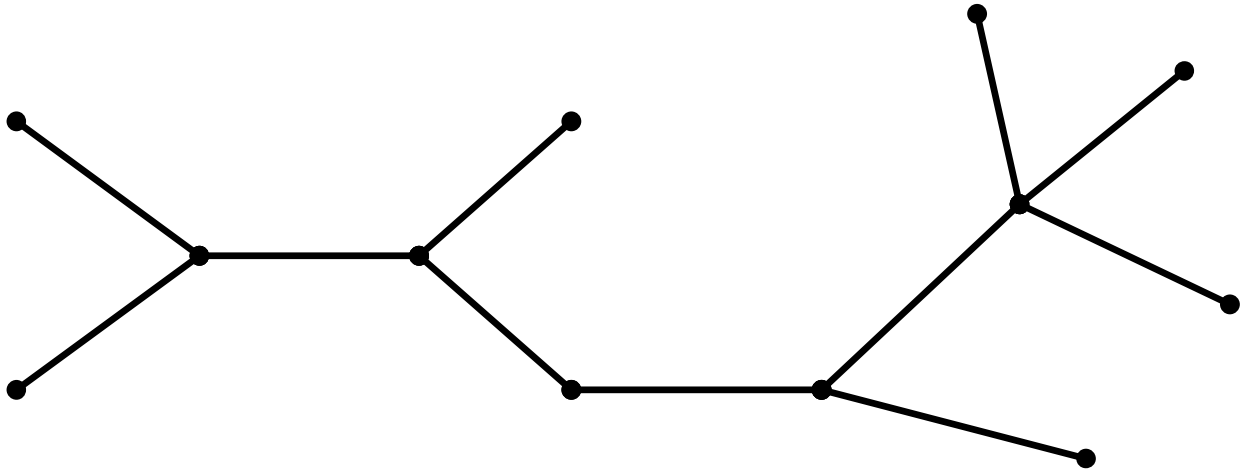


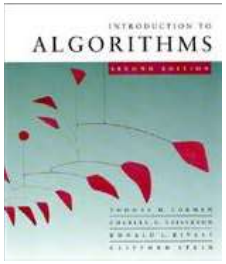


Optimal substructure

MST T:

(Other edges of G are not shown.)

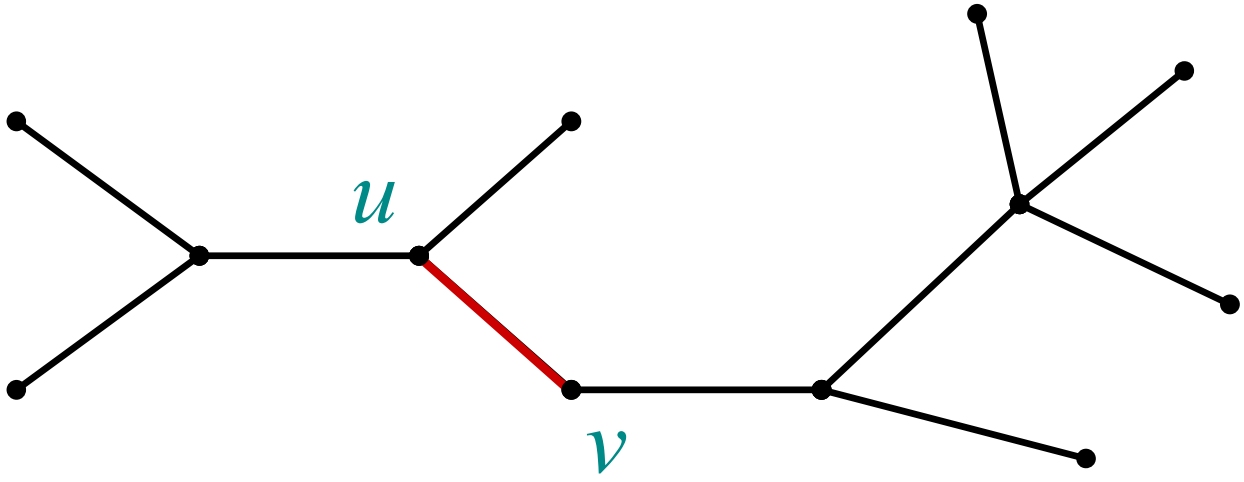




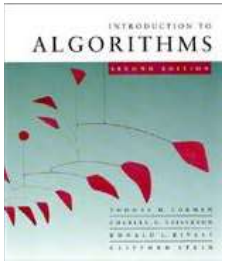
Optimal substructure

MST T:

(Other edges of G are not shown.)



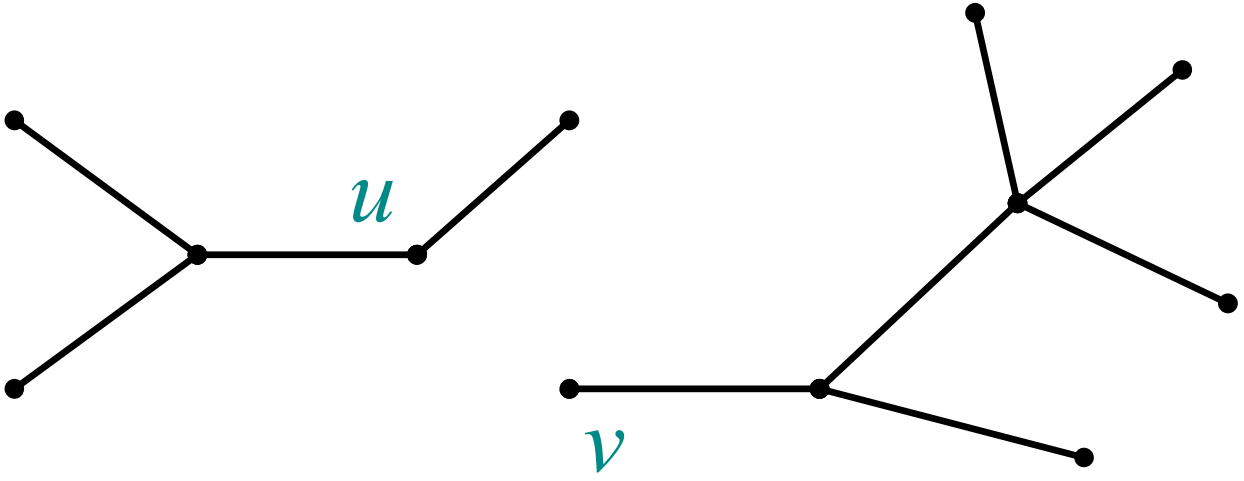
Remove any edge $(u, v) \in T$.



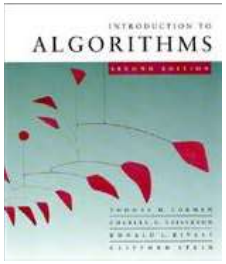
Optimal substructure

MST T:

(Other edges of G are not shown.)



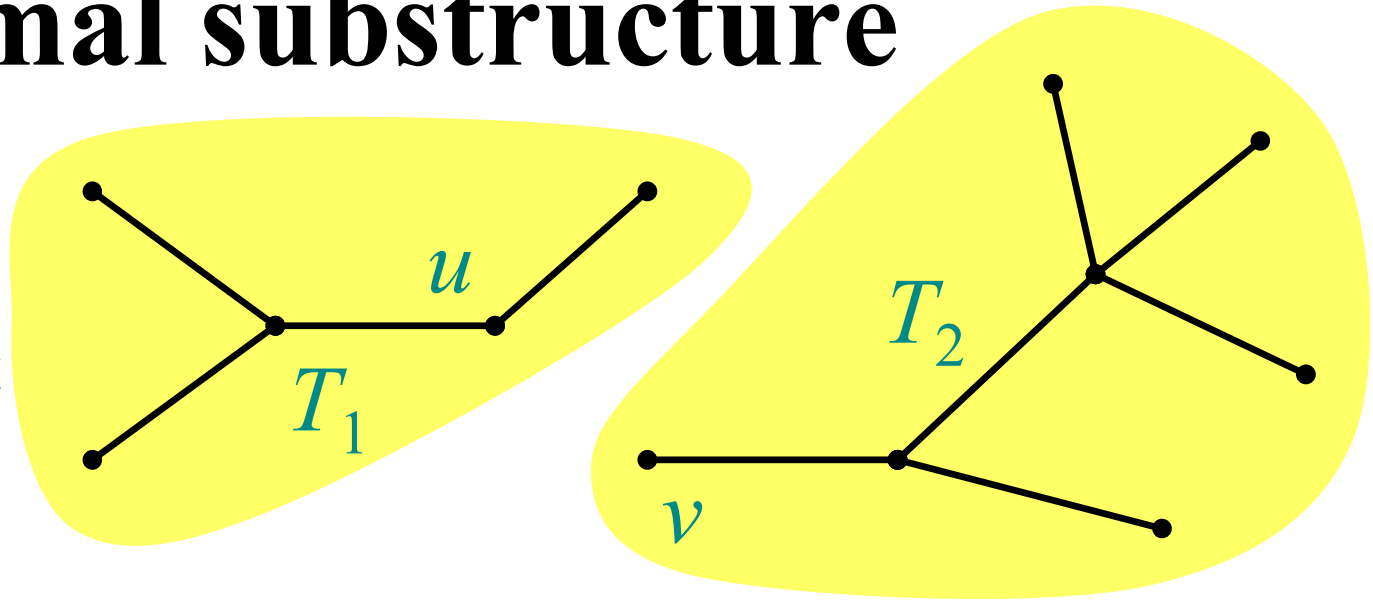
Remove any edge $(u, v) \in T$.



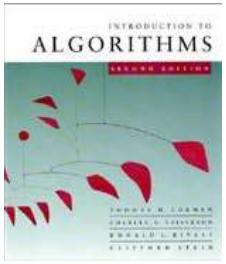
Optimal substructure

MST T:

(Other edges of G are not shown.)



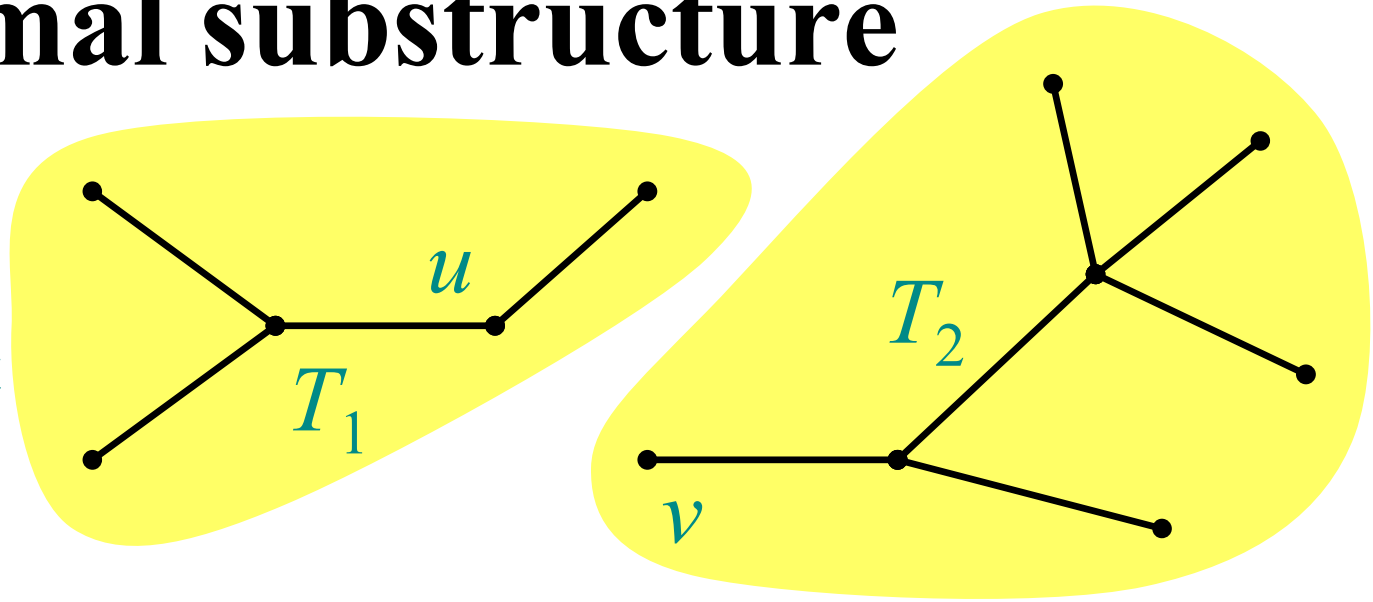
Remove any edge $(u, v) \in T$. Then, T is partitioned into two subtrees T_1 and T_2.



Optimal substructure

MST T:

(Other edges of G are not shown.)

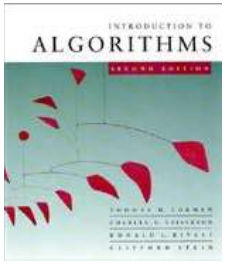


Remove any edge $(u, v) \in T$. Then, T is partitioned into two subtrees T_1 and T_2.

Theorem. The subtree T_1 is an MST of $G_1 = (V_1, E_1)$, the subgraph of G ***induced*** by the vertices of T_1:

$$V_1 = \text{vertices of } T_1,$$
$$E_1 = \{ (x, y) \in E : x, y \in V_1 \}.$$

Similarly for T_2.

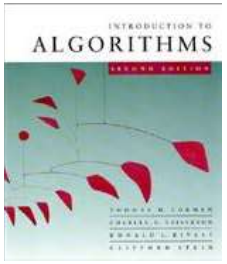


Proof of optimal substructure

Proof. Cut and paste:

$$w(T) = w(u, v) + w(T_1) + w(T_2).$$

If T_1' were a lower-weight spanning tree than T_1 for G_1, then $T' = \{(u, v)\} \cup T_1' \cup T_2$ would be a lower-weight spanning tree than T for G. □



Proof of optimal substructure

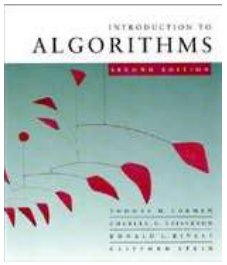
Proof. Cut and paste:

$$w(T) = w(u, v) + w(T_1) + w(T_2).$$

If T_1' were a lower-weight spanning tree than T_1 for G_1, then $T' = \{(u, v)\} \cup T_1' \cup T_2$ would be a lower-weight spanning tree than T for G. □

Do we also have overlapping subproblems?

- Yes.



Proof of optimal substructure

Proof. Cut and paste:

$$w(T) = w(u, v) + w(T_1) + w(T_2).$$

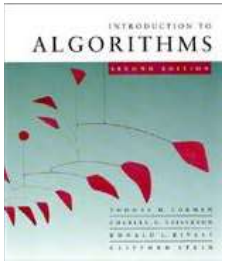
If T_1' were a lower-weight spanning tree than T_1 for G_1, then $T' = \{(u, v)\} \cup T_1' \cup T_2$ would be a lower-weight spanning tree than T for G. □

Do we also have overlapping subproblems?

- Yes.

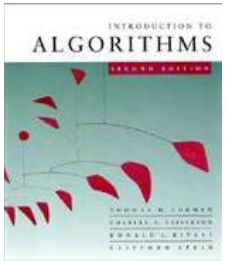
Great, then dynamic programming may work!

- Yes, but MST exhibits another powerful property which leads to an even more efficient algorithm.



Hallmark for "greedy" algorithms

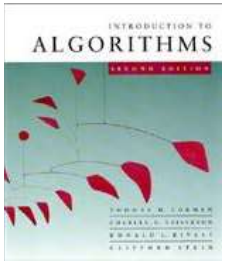
Greedy-choice property
A locally optimal choice is globally optimal.



Hallmark for "greedy" algorithms

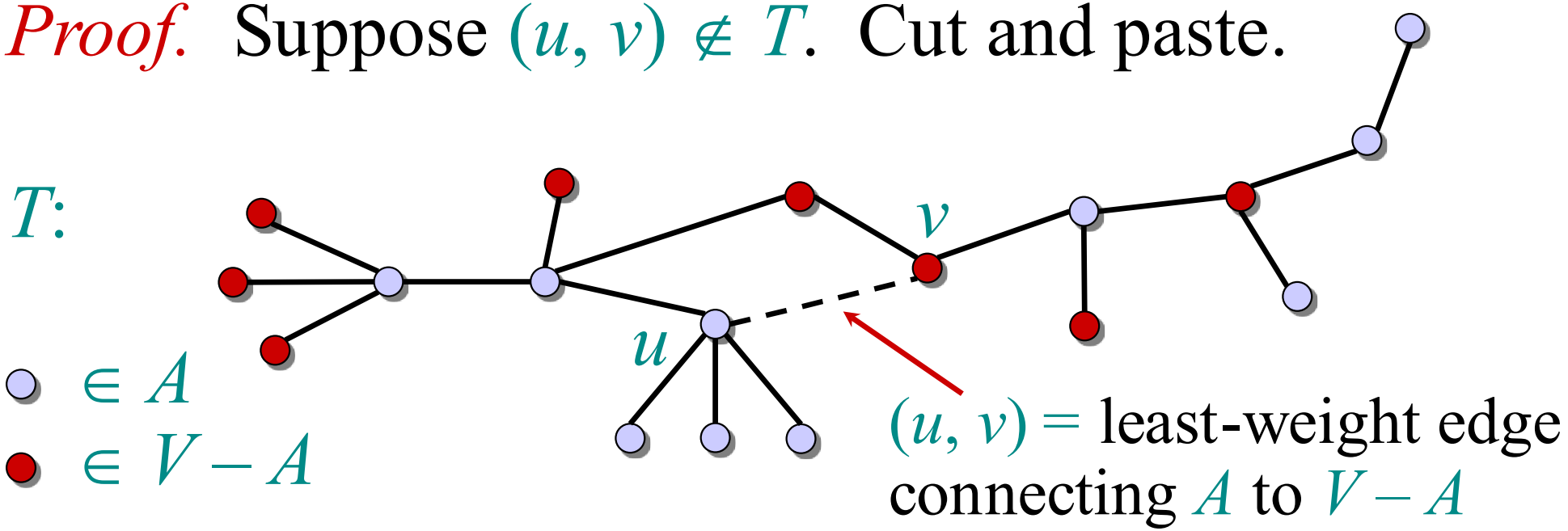
Greedy-choice property
A locally optimal choice is globally optimal.

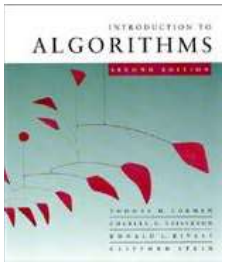
Theorem. Let T be the MST of $G = (V, E)$, and let $A \subseteq V$. Suppose that $(u, v) \in E$ is the least-weight edge connecting A to $V - A$. Then, $(u, v) \in T$.



Proof of theorem

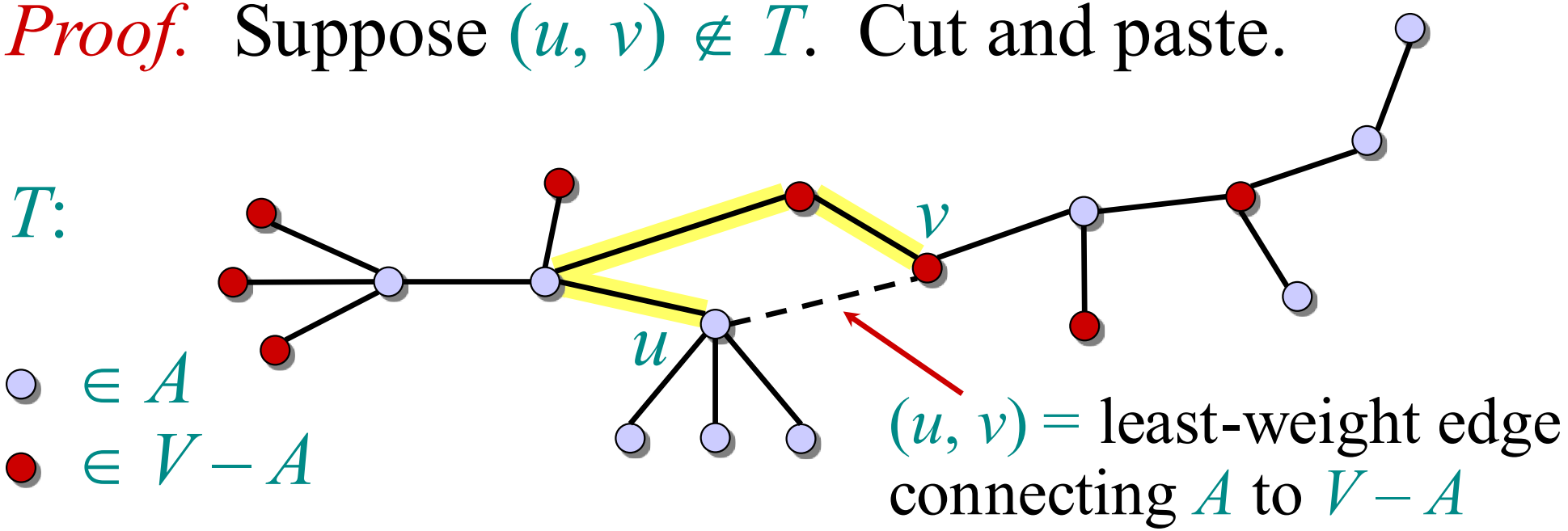
Proof. Suppose $(u, v) \notin T$. Cut and paste.



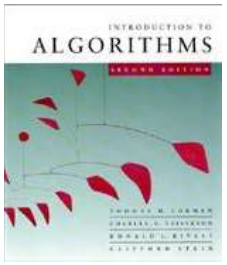


Proof of theorem

Proof. Suppose $(u, v) \notin T$. Cut and paste.

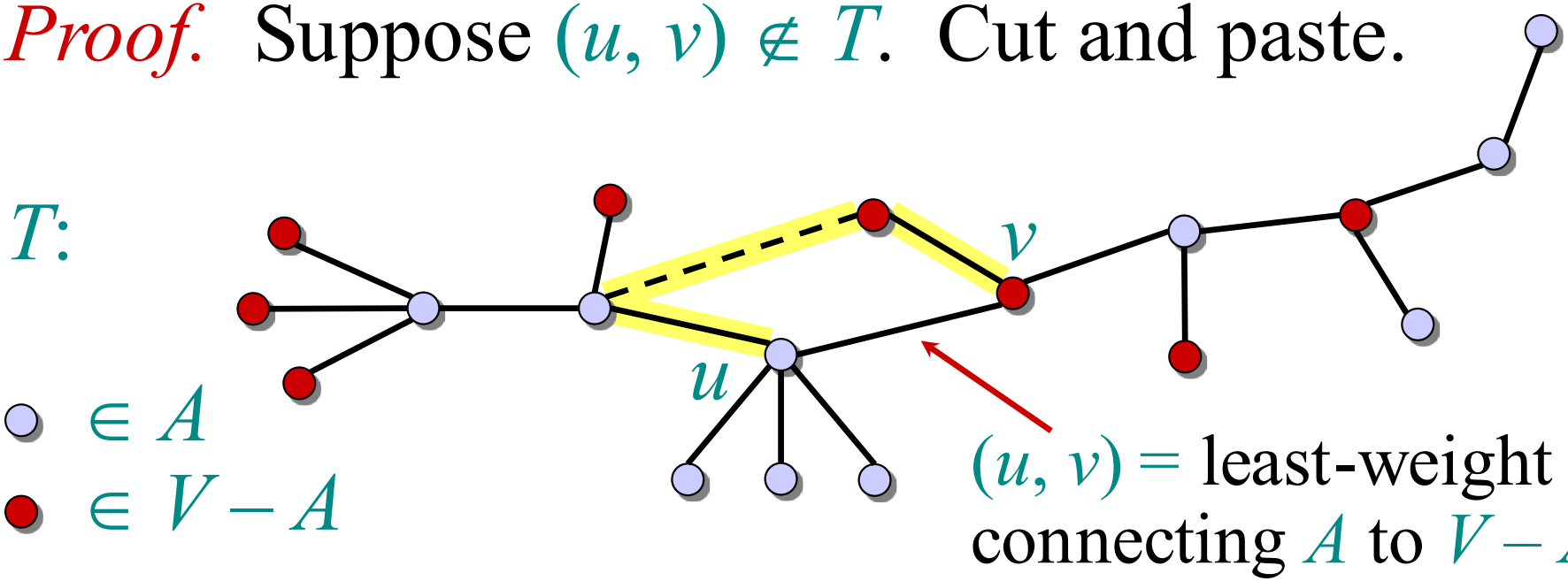


Consider the unique simple path from u to v in T.



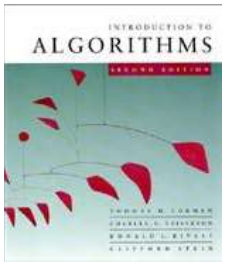
Proof of theorem

Proof. Suppose $(u, v) \notin T$. Cut and paste.



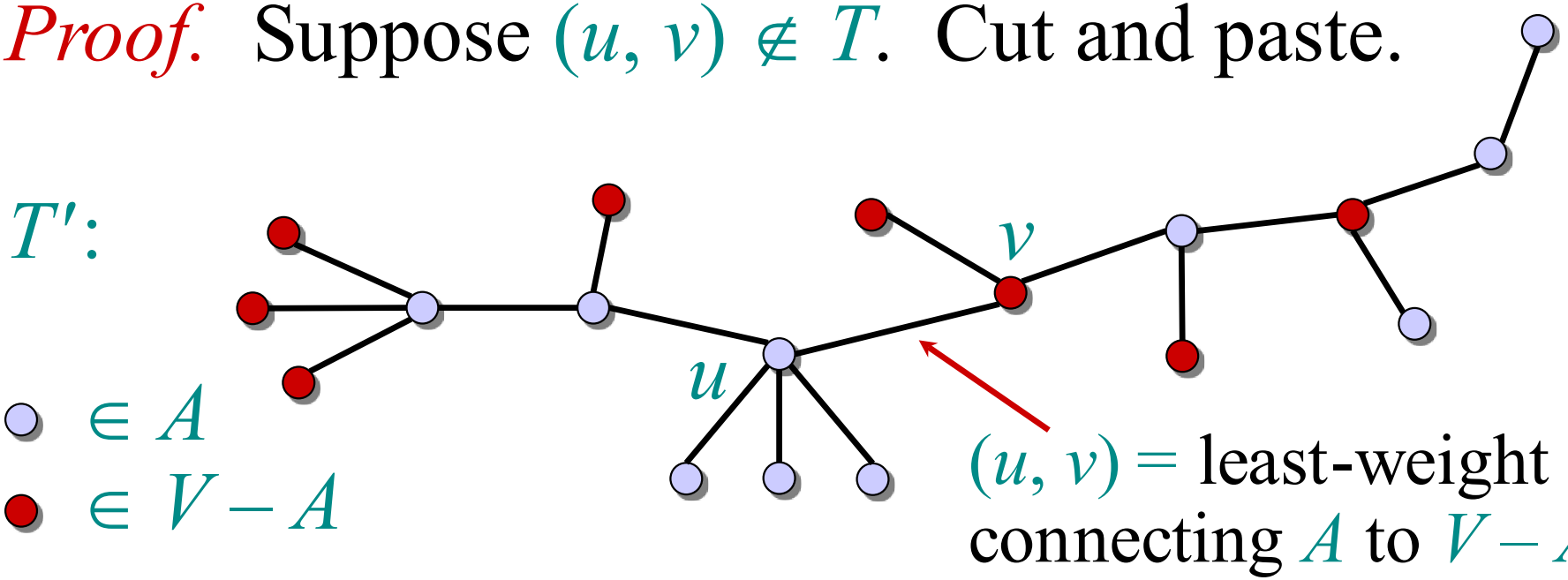
Consider the unique simple path from u to v in T.

Swap (u, v) with the first edge on this path that connects a vertex in A to a vertex in $V - A$.



Proof of theorem

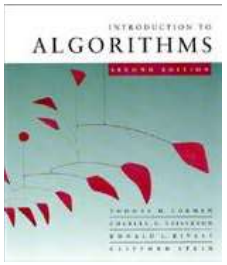
Proof. Suppose $(u, v) \notin T$. Cut and paste.



Consider the unique simple path from u to v in T.

Swap (u, v) with the first edge on this path that connects a vertex in A to a vertex in $V - A$.

A lighter-weight spanning tree than T results. □

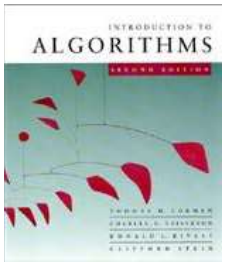


Prim's algorithm

IDEA: Maintain $V - A$ as a priority queue Q. Key each vertex in Q with the weight of the least-weight edge connecting it to a vertex in A.

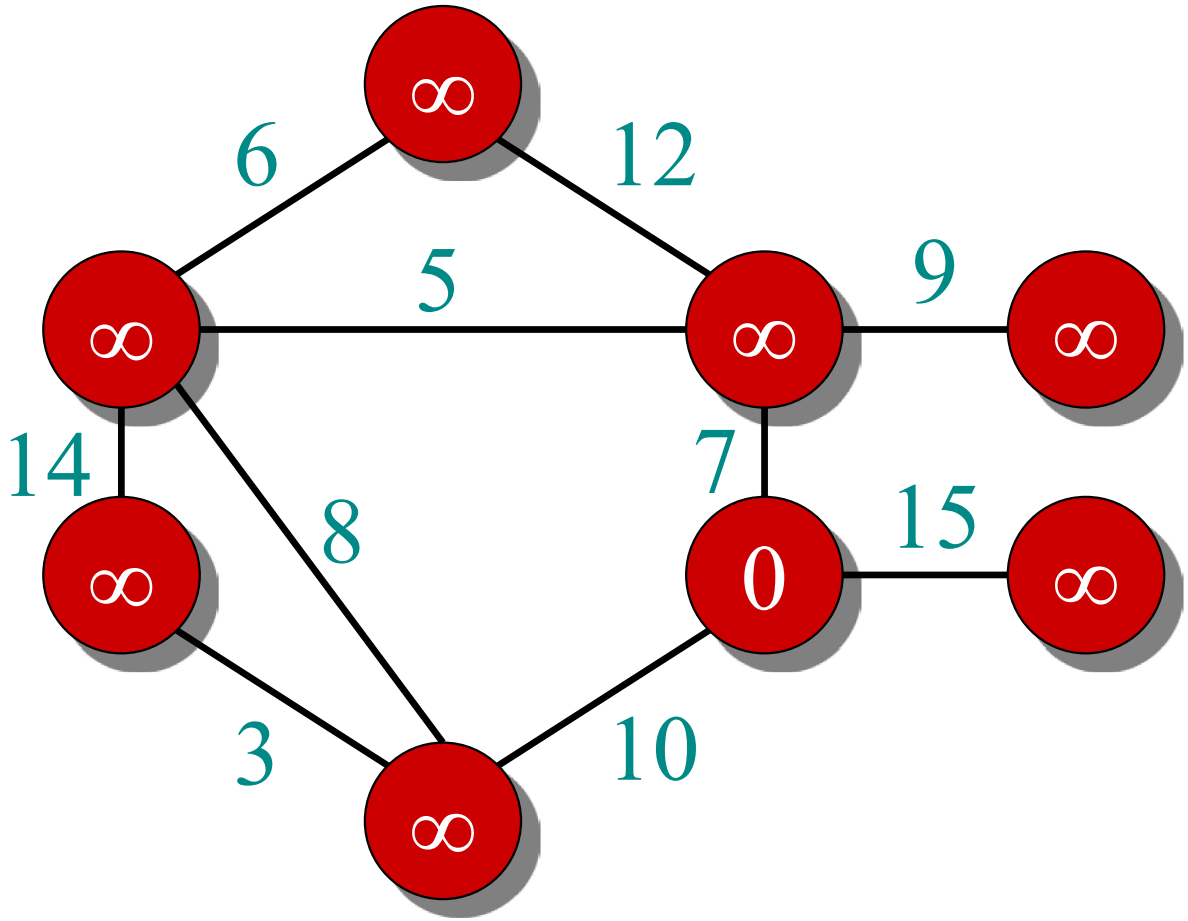
$Q \leftarrow V$
$key[v] \leftarrow \infty$ for all $v \in V$
$key[s] \leftarrow 0$ for some arbitrary $s \in V$
while $Q \neq \varnothing$
    do $u \leftarrow$ EXTRACT-MIN(Q)
        for each $v \in Adj[u]$
            do if $v \in Q$ and $w(u, v) < key[v]$
                then $key[v] \leftarrow w(u, v)$ ▷ DECREASE-KEY
                $\pi[v] \leftarrow u$

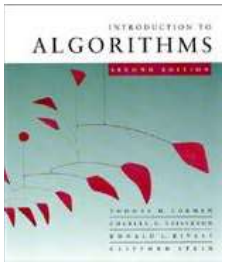
At the end, $\{(v, \pi[v])\}$ forms the MST.



Example of Prim's algorithm

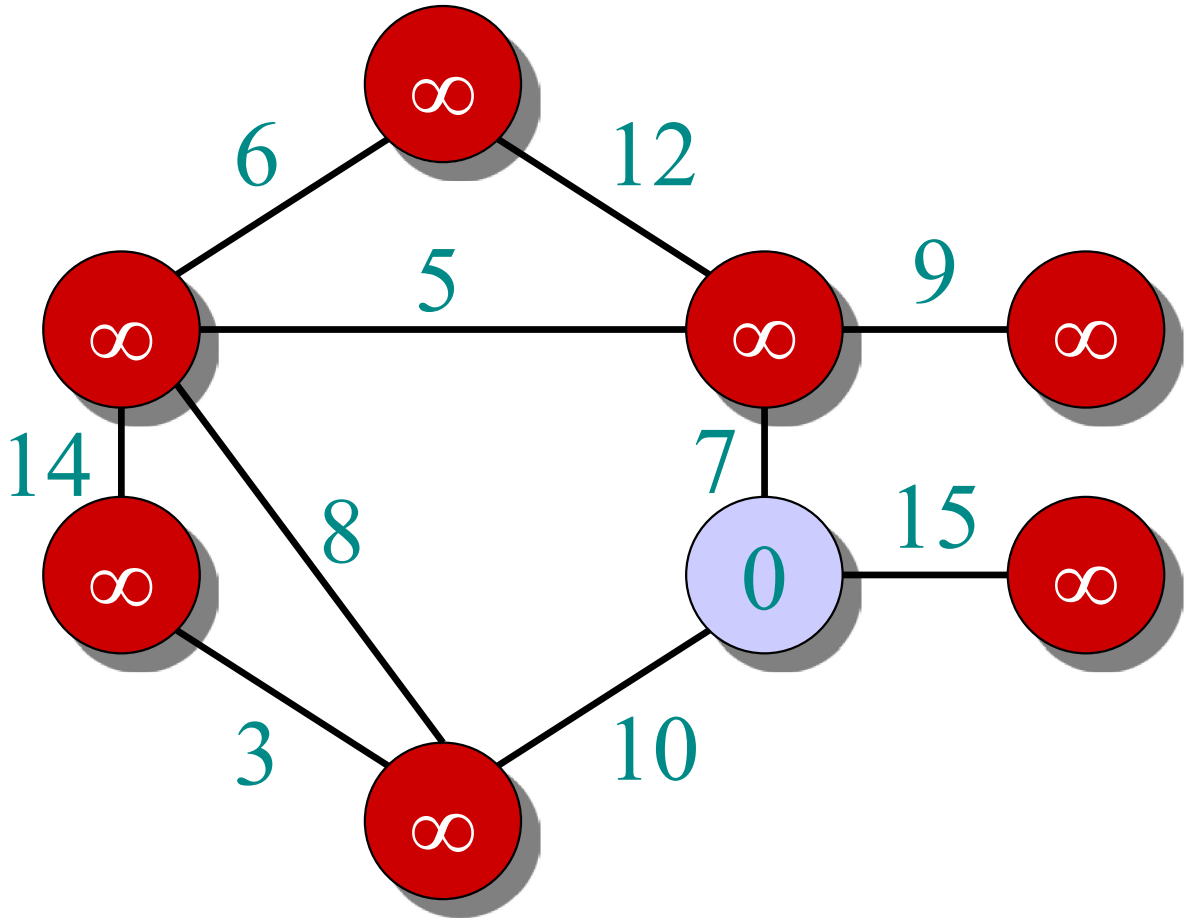
- $\in A$
- $\in V - A$

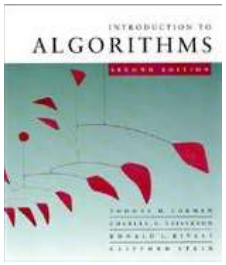




Example of Prim's algorithm

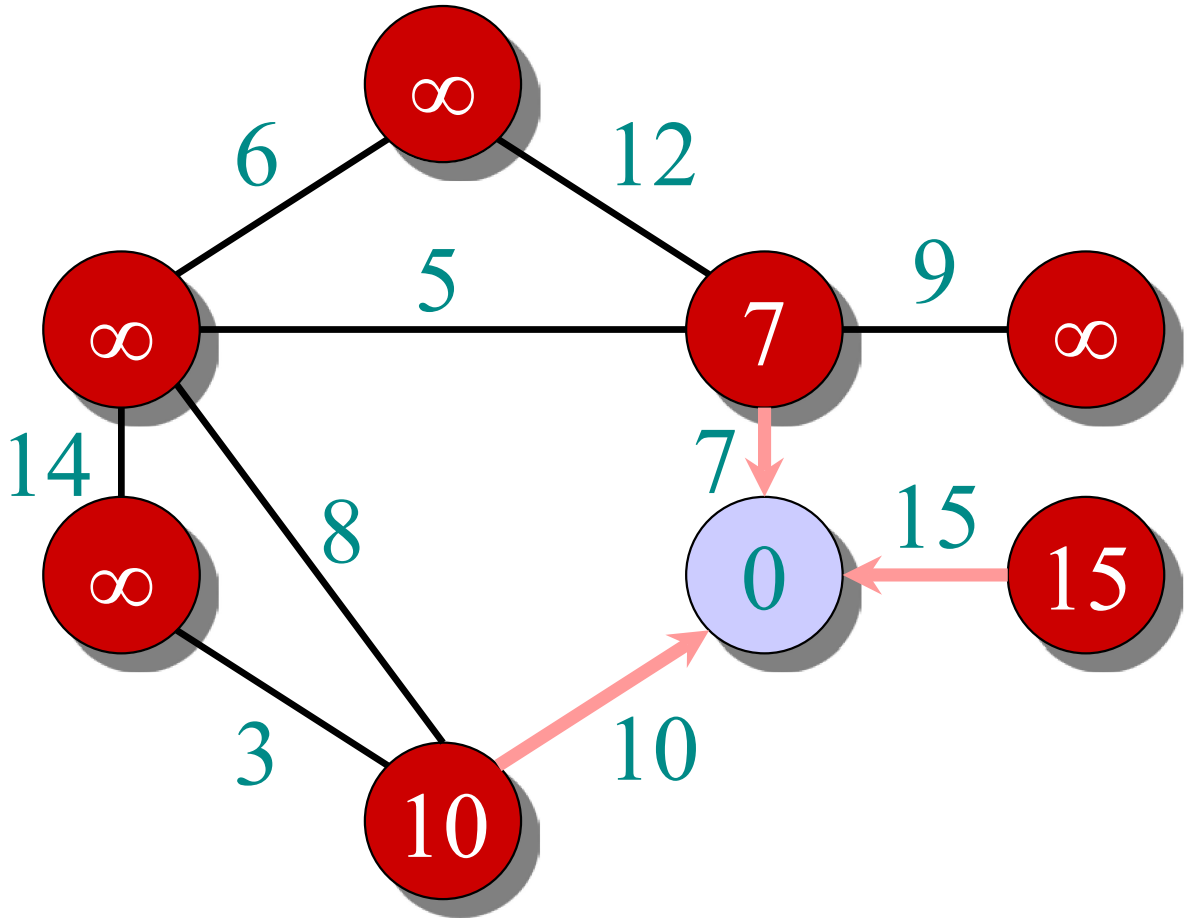
- $\in A$
- $\in V - A$

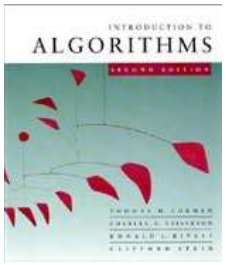




Example of Prim's algorithm

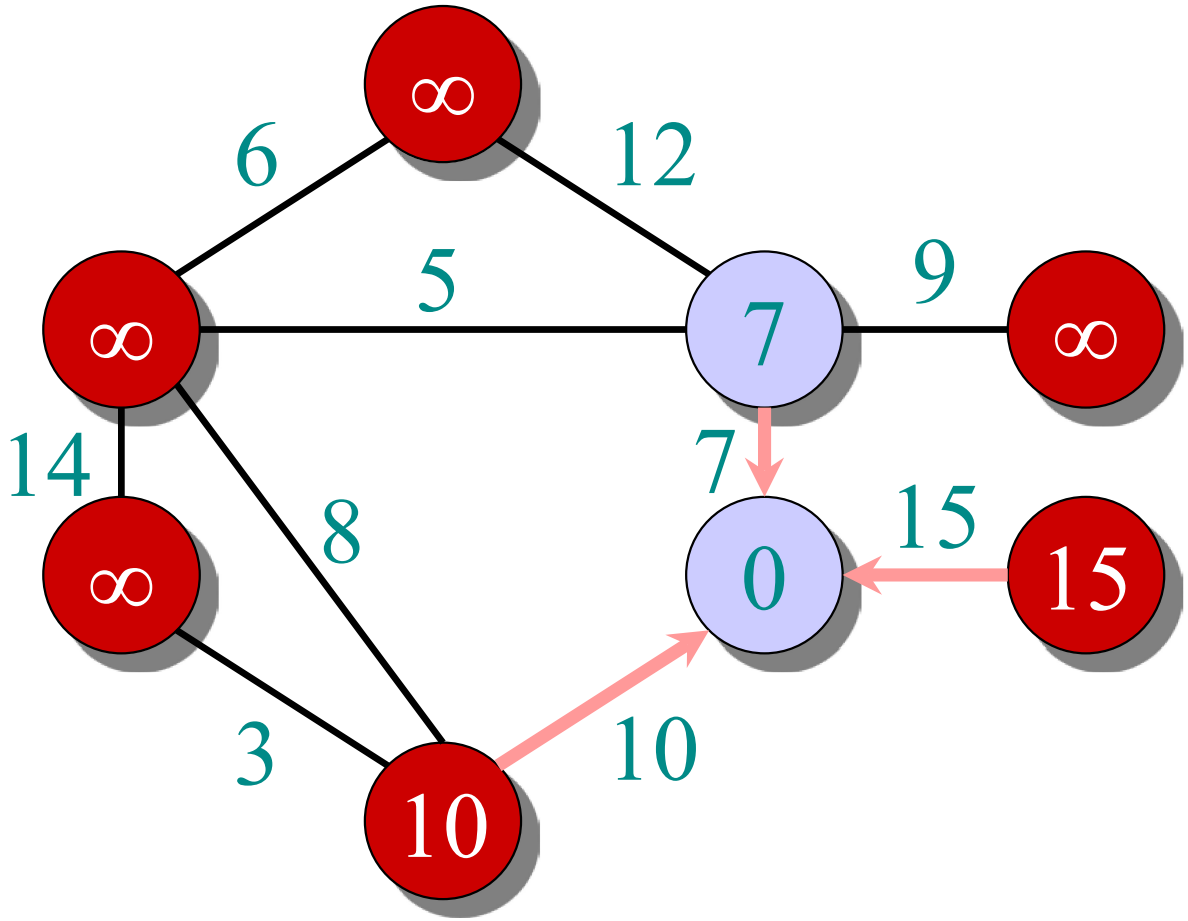
- $\in A$
- $\in V - A$

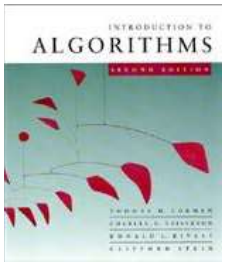




Example of Prim's algorithm

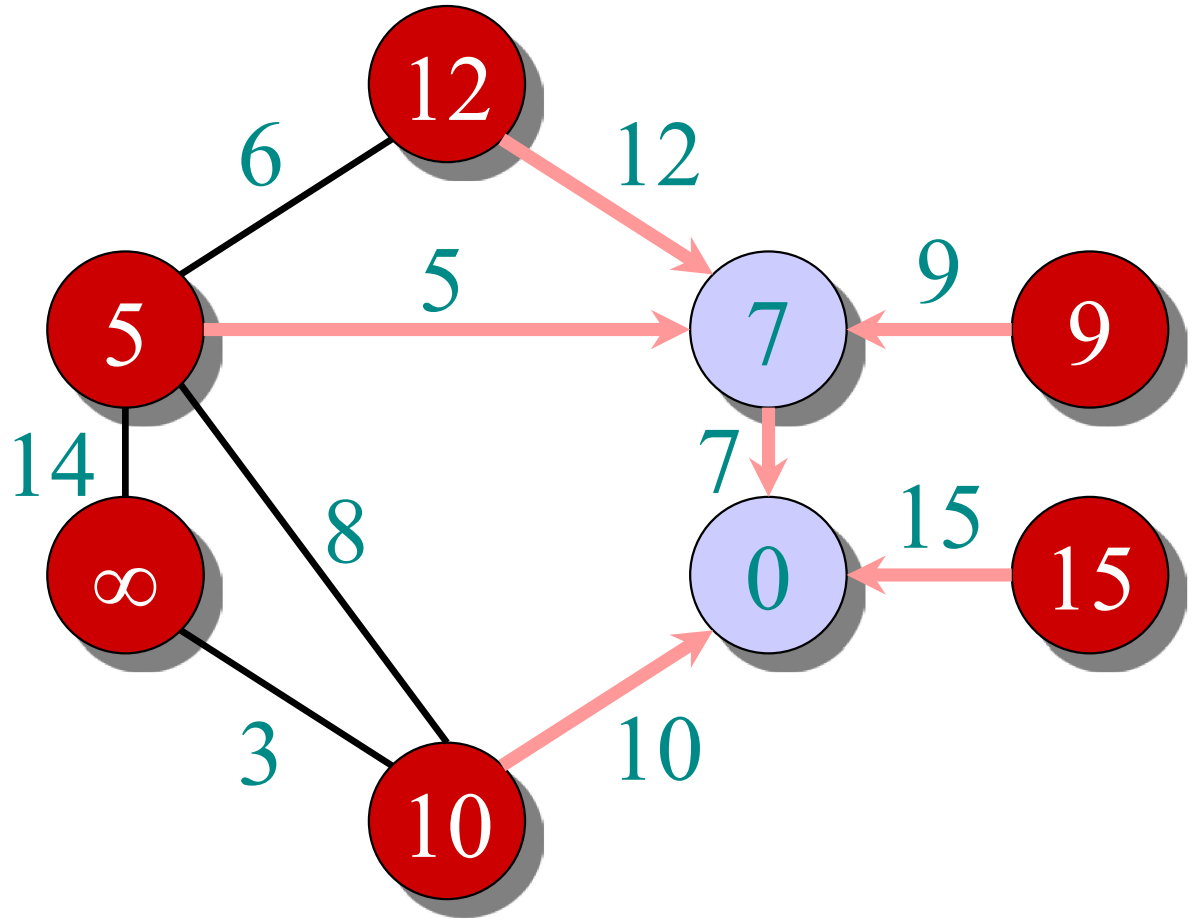
- $\in A$
- $\in V - A$

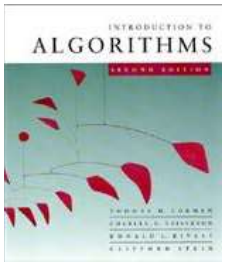




Example of Prim's algorithm

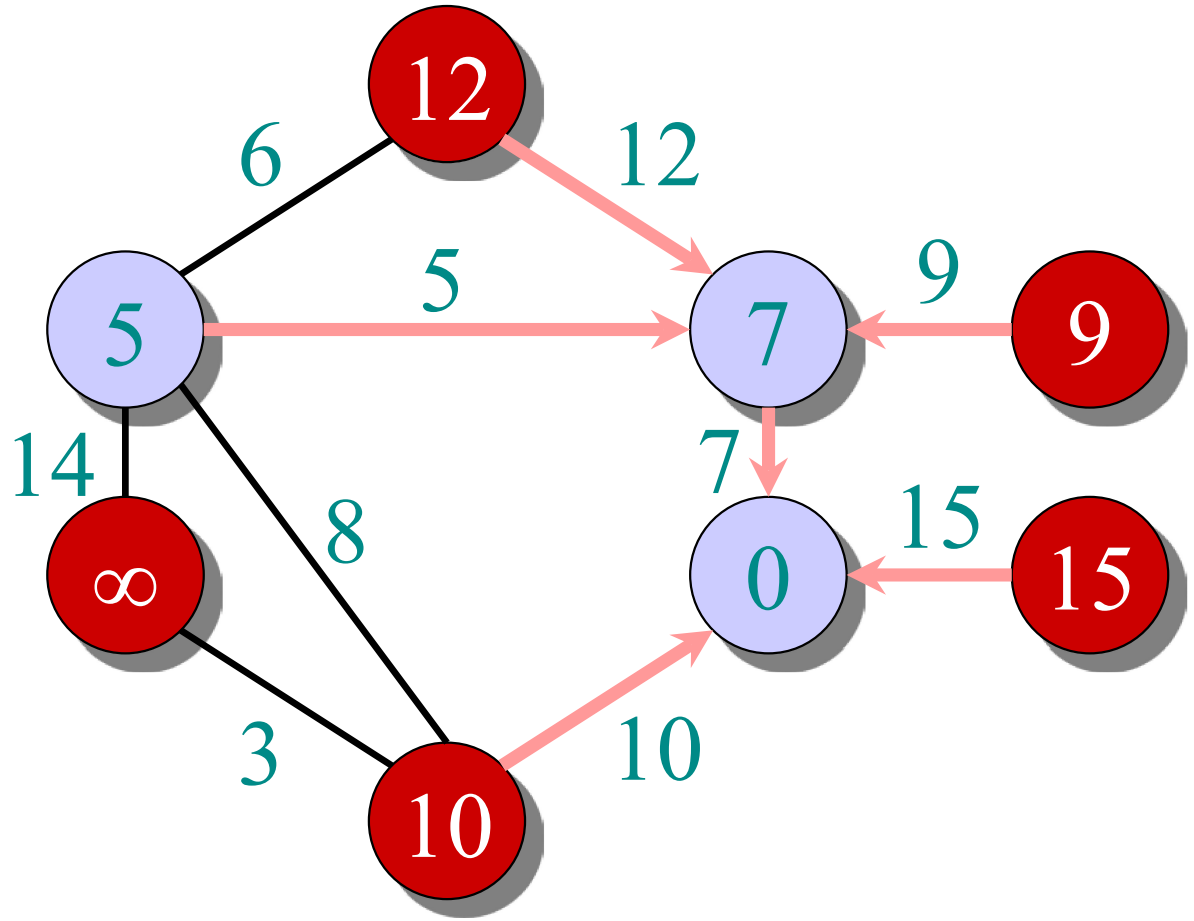
- $\in A$
- $\in V - A$

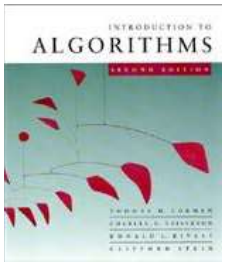




Example of Prim's algorithm

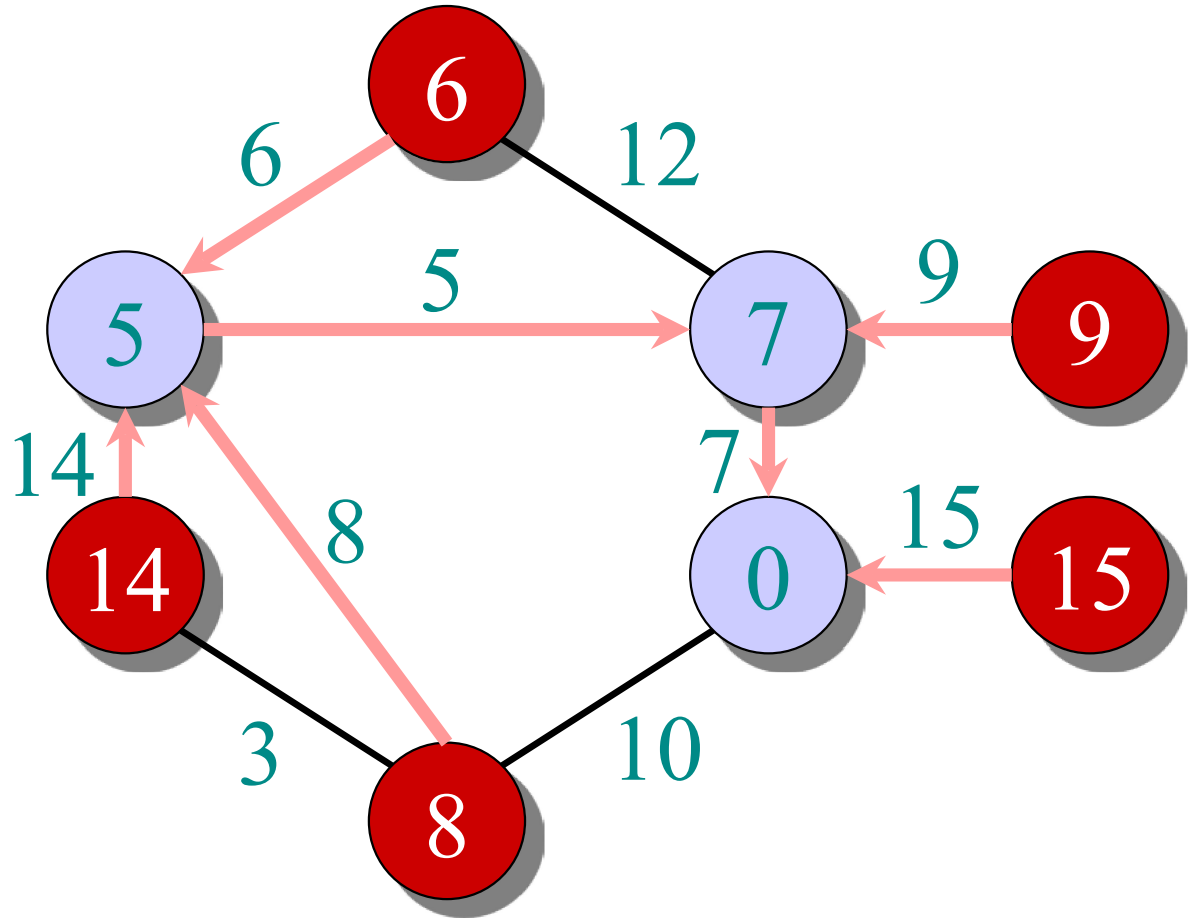
- $\in A$
- $\in V - A$

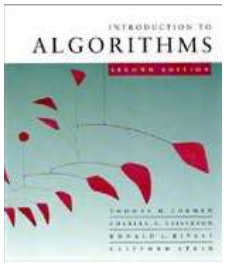




Example of Prim's algorithm

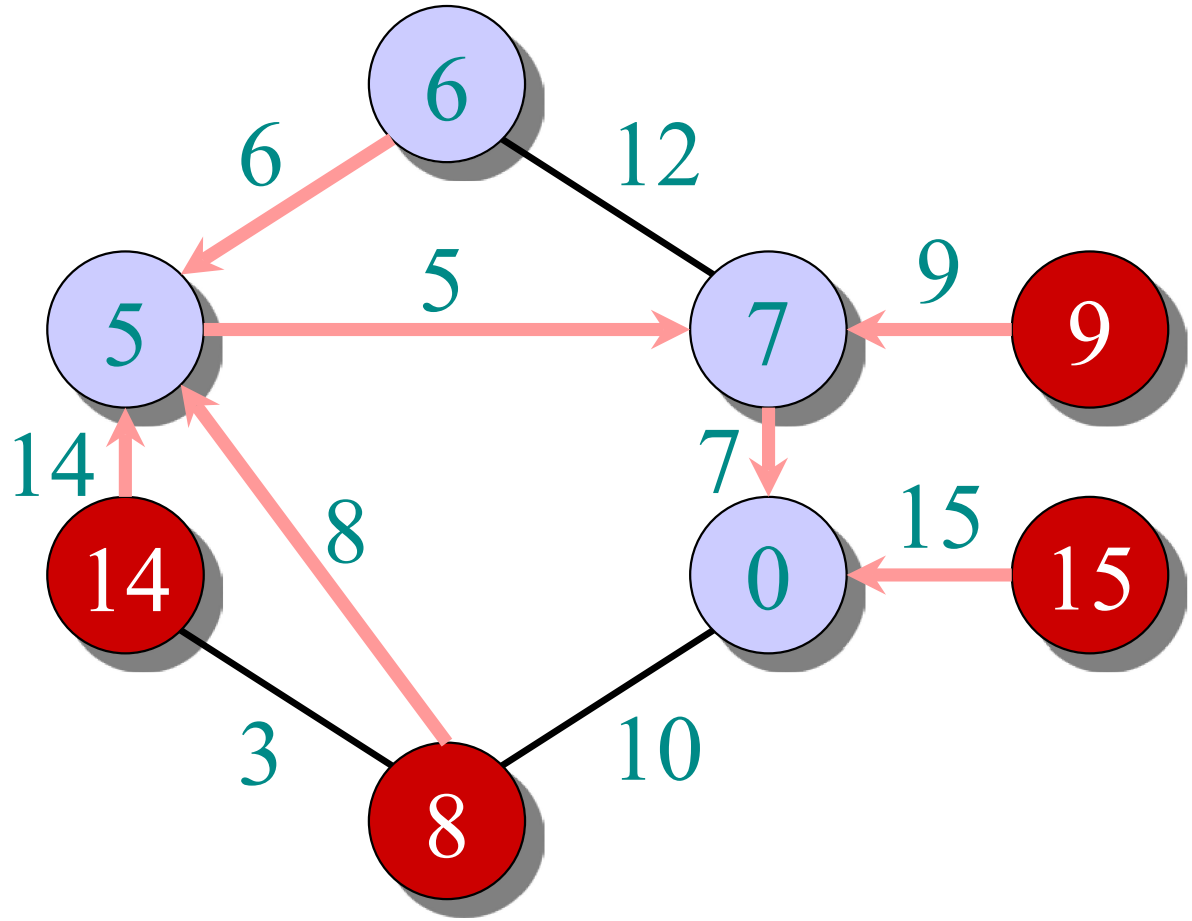
- $\in A$
- $\in V - A$

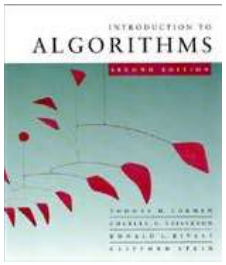




Example of Prim's algorithm

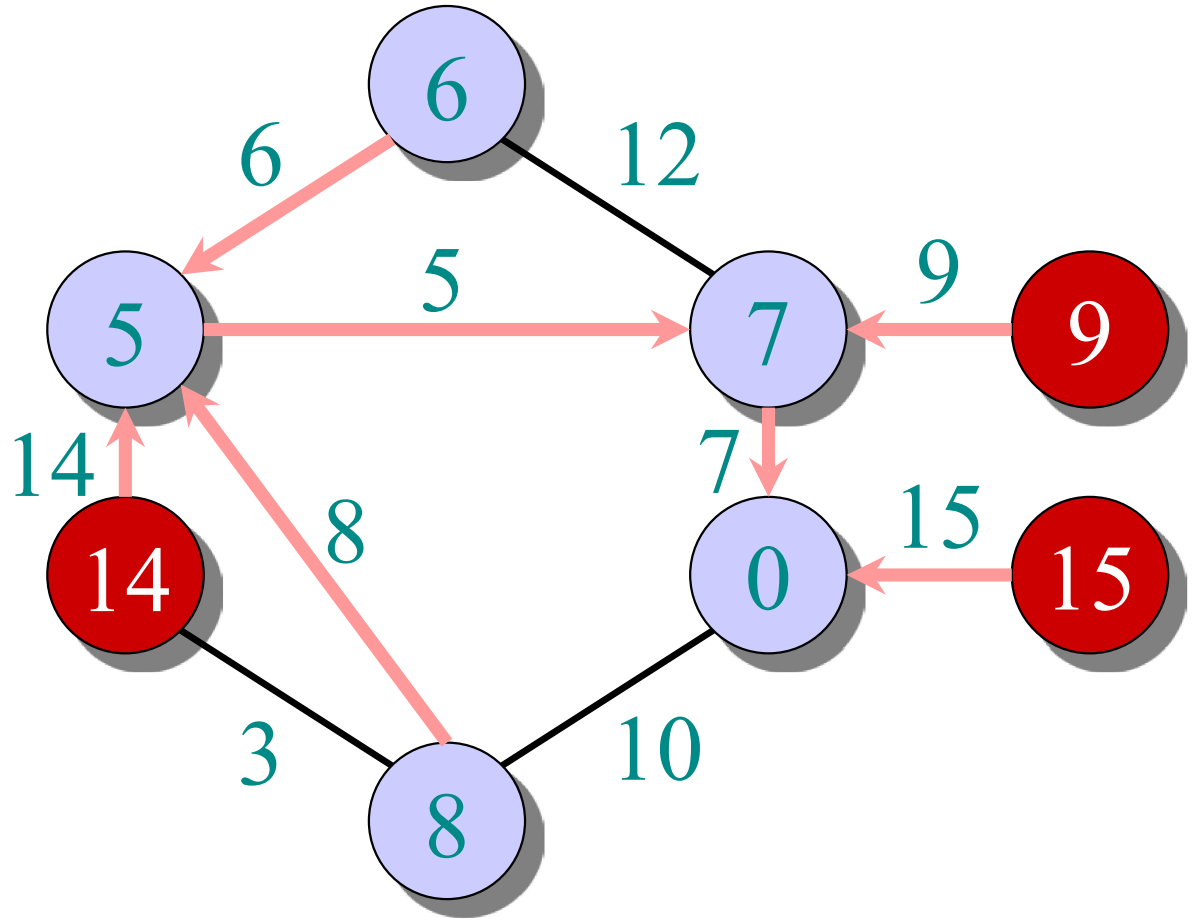
- $\in A$
- $\in V - A$

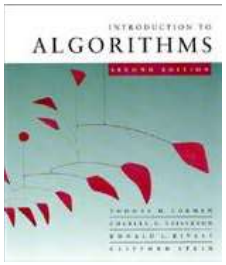




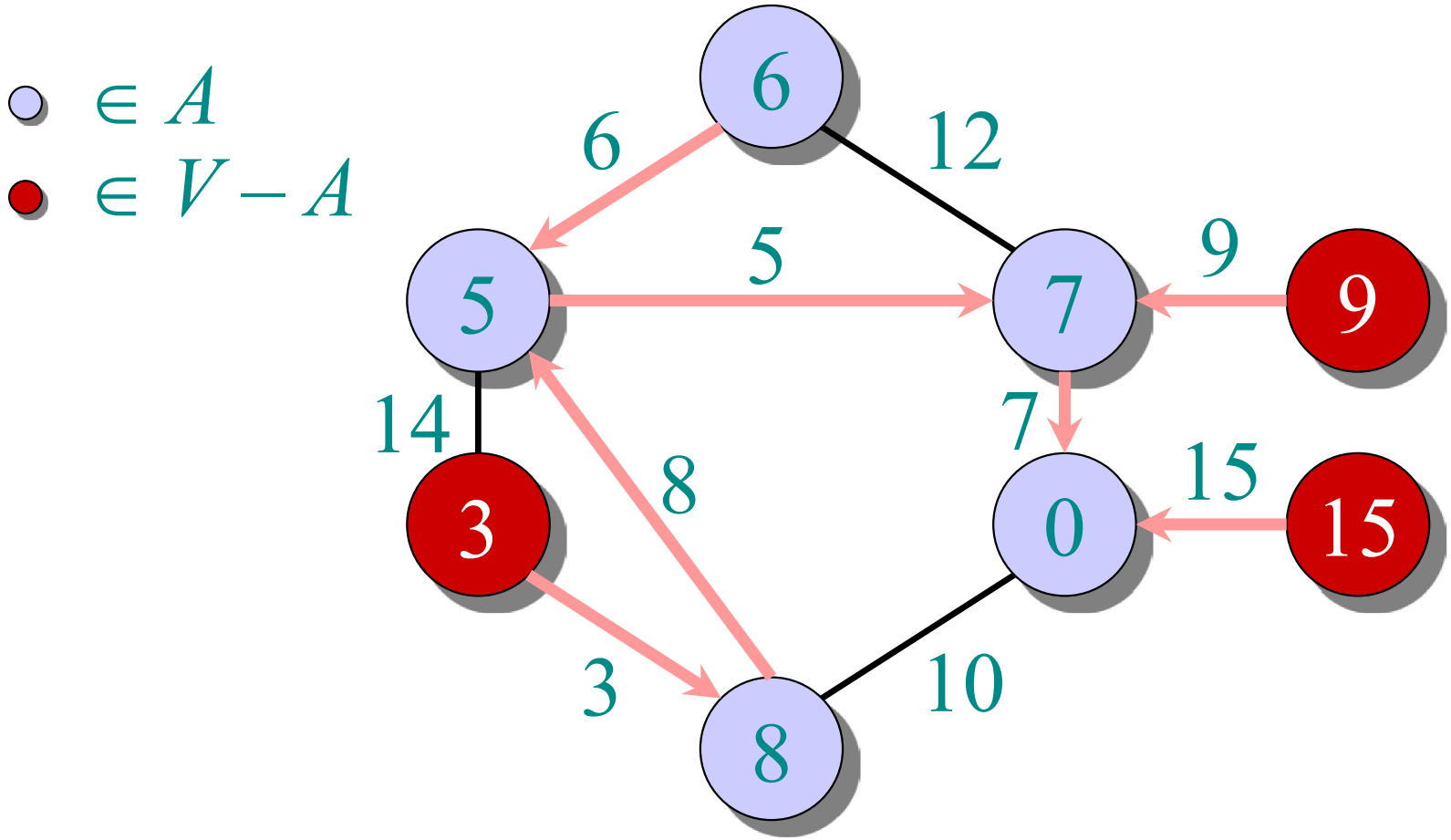
Example of Prim's algorithm

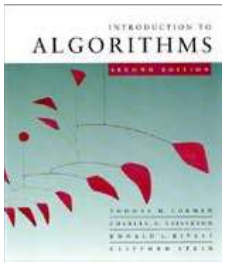
- $\in A$
- $\in V - A$





Example of Prim's algorithm

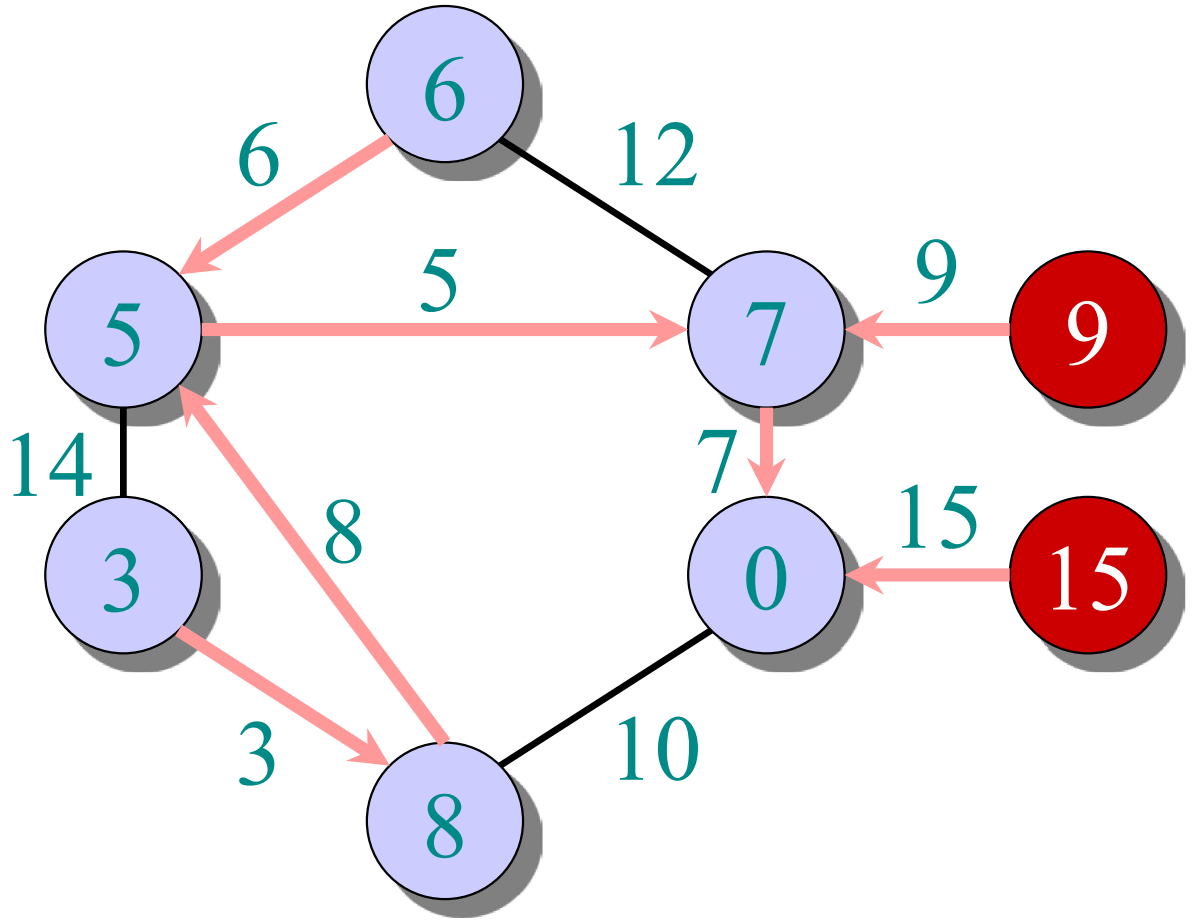


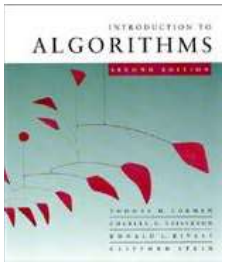


Example of Prim's algorithm

$\in A$

$\in V - A$

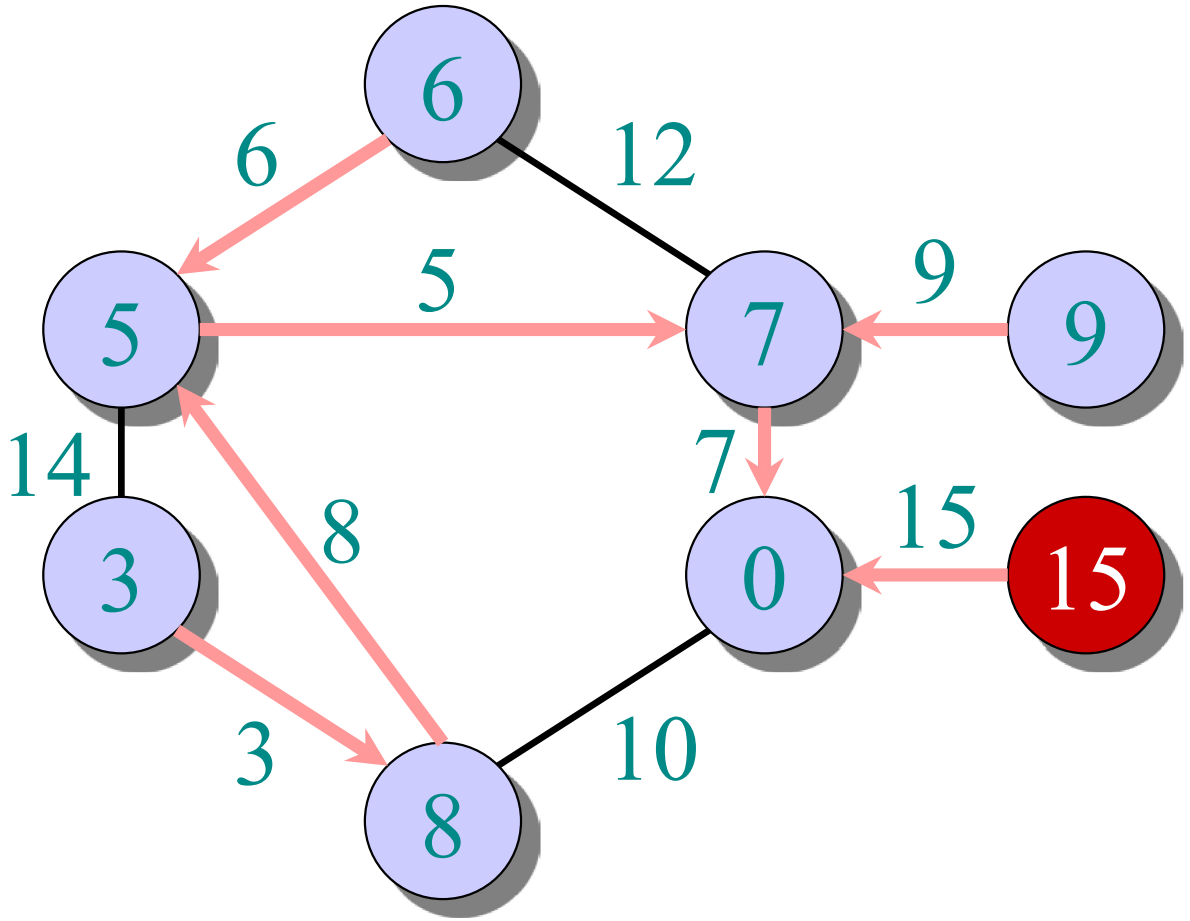


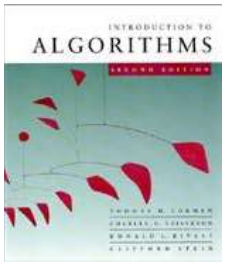


Example of Prim's algorithm

$\in A$

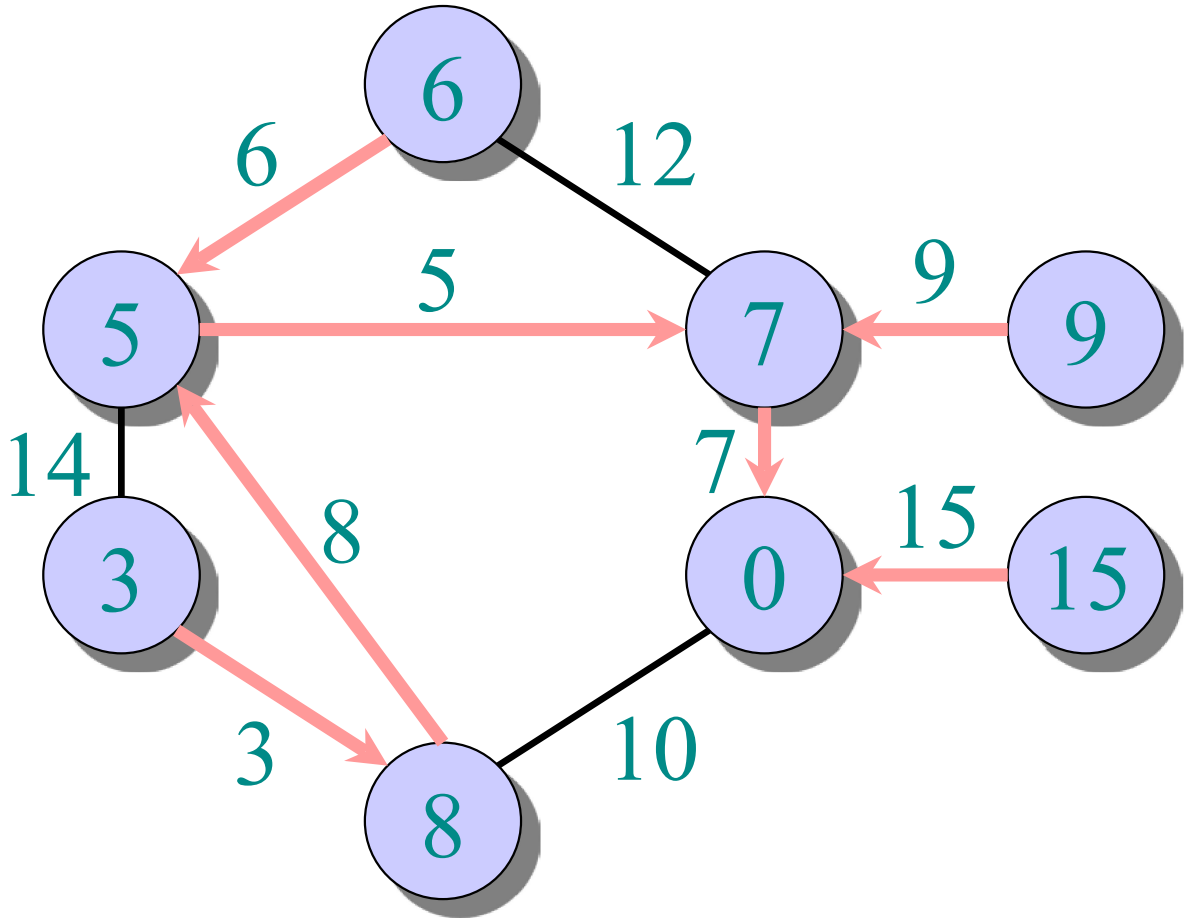
$\in V - A$

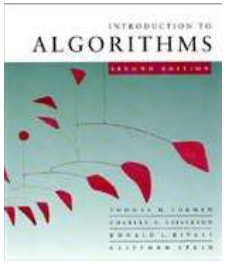




Example of Prim's algorithm

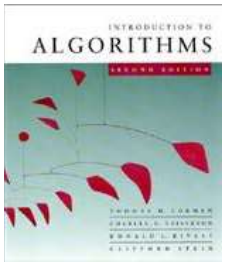
- $\in A$
- $\in V - A$





Analysis of Prim

$Q \leftarrow V$
$key[v] \leftarrow \infty$ for all $v \in V$
$key[s] \leftarrow 0$ for some arbitrary $s \in V$
while $Q \neq \varnothing$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    for each $v \in Adj[u]$
      do if $v \in Q$ and $w(u, v) < key[v]$
        then $key[v] \leftarrow w(u, v)$
          $\pi[v] \leftarrow u$



Analysis of Prim

$\Theta(V)$ total

$Q \leftarrow V$

$key[v] \leftarrow \infty$ for all $v \in V$

$key[s] \leftarrow 0$ for some arbitrary $s \in V$

while $Q \neq \varnothing$

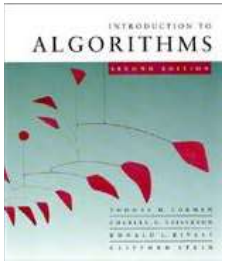
  do $u \leftarrow$ EXTRACT-MIN(Q)

    for each $v \in Adj[u]$

      do if $v \in Q$ and $w(u, v) < key[v]$

        then $key[v] \leftarrow w(u, v)$

          $\pi[v] \leftarrow u$



Analysis of Prim

$\Theta(V)$ total

$Q \leftarrow V$

$key[v] \leftarrow \infty$ for all $v \in V$

$key[s] \leftarrow 0$ for some arbitrary $s \in V$

while $Q \neq \varnothing$

$|V|$ times

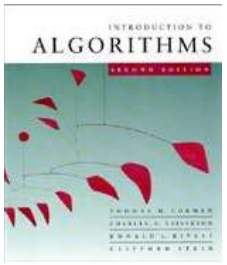
do $u \leftarrow$ EXTRACT-MIN(Q)

for each $v \in Adj[u]$

do if $v \in Q$ and $w(u, v) < key[v]$

then $key[v] \leftarrow w(u, v)$

$\pi[v] \leftarrow u$



Analysis of Prim

$\Theta(V)$ total:
- $Q \leftarrow V$
- $key[v] \leftarrow \infty$ for all $v \in V$
- $key[s] \leftarrow 0$ for some arbitrary $s \in V$

while $Q \neq \varnothing$ — $|V|$ times

  do $u \leftarrow$ EXTRACT-MIN(Q)

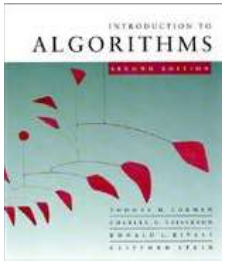
  $degree(u)$ times:

  for each $v \in Adj[u]$

    do if $v \in Q$ and $w(u, v) < key[v]$

      then $key[v] \leftarrow w(u, v)$

        $\pi[v] \leftarrow u$



Analysis of Prim

$\Theta(V)$ total:
- $Q \leftarrow V$
- $key[v] \leftarrow \infty$ for all $v \in V$
- $key[s] \leftarrow 0$ for some arbitrary $s \in V$

while $Q \neq \varnothing$ — $|V|$ times

  do $u \leftarrow$ EXTRACT-MIN(Q)

  $degree(u)$ times:

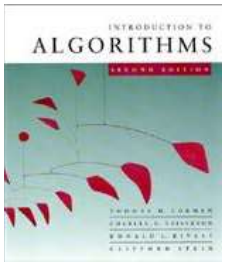
  for each $v \in Adj[u]$

    do if $v \in Q$ and $w(u, v) < key[v]$

      then $key[v] \leftarrow w(u, v)$

        $\pi[v] \leftarrow u$

Handshaking Lemma $\Rightarrow$ $\Theta(E)$ implicit DECREASE-KEY's.



Analysis of Prim

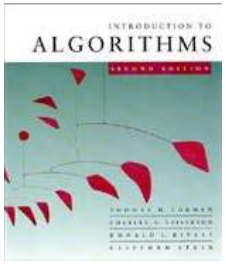
$\Theta(V)$ total:
- $Q \leftarrow V$
- $key[v] \leftarrow \infty$ for all $v \in V$
- $key[s] \leftarrow 0$ for some arbitrary $s \in V$

$|V|$ times:
- **while** $Q \neq \varnothing$
- **do** $u \leftarrow$ EXTRACT-MIN(Q)

$degree(u)$ times:
- **for** each $v \in Adj[u]$
- **do if** $v \in Q$ and $w(u, v) < key[v]$
- **then** $key[v] \leftarrow w(u, v)$
- $\pi[v] \leftarrow u$

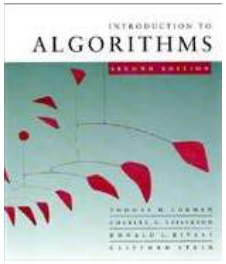
Handshaking Lemma $\Rightarrow \Theta(E)$ implicit DECREASE-KEY's.

$$\text{Time} = \Theta(V) \cdot T_{\text{EXTRACT-MIN}} + \Theta(E) \cdot T_{\text{DECREASE-KEY}}$$



Analysis of Prim (continued)

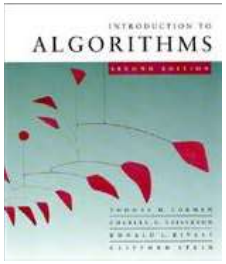
$$\text{Time} = \Theta(V)\cdot T_{\text{EXTRACT-MIN}} + \Theta(E)\cdot T_{\text{DECREASE-KEY}}$$



Analysis of Prim (continued)

$$\text{Time} = \Theta(V)\cdot T_{\text{EXTRACT-MIN}} + \Theta(E)\cdot T_{\text{DECREASE-KEY}}$$

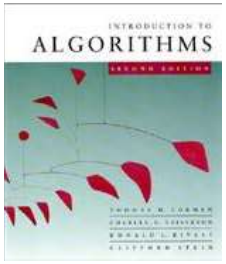
Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total



Analysis of Prim (continued)

$$\text{Time} = \Theta(V)\cdot T_{\text{EXTRACT-MIN}} + \Theta(E)\cdot T_{\text{DECREASE-KEY}}$$

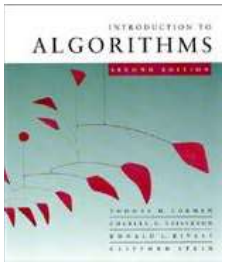
Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$



Analysis of Prim (continued)

$$\text{Time} = \Theta(V)\cdot T_{\text{EXTRACT-MIN}} + \Theta(E)\cdot T_{\text{DECREASE-KEY}}$$

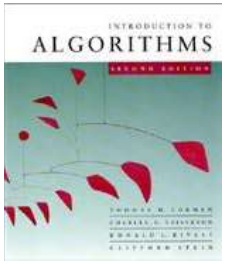
Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$



Analysis of Prim (continued)

$$\text{Time} = \Theta(V)\cdot T_{\text{EXTRACT-MIN}} + \Theta(E)\cdot T_{\text{DECREASE-KEY}}$$

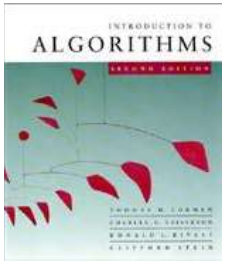
Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap	$O(\lg V)$ amortized	$O(1)$ amortized	$O(E + V \lg V)$ worst case



MST algorithms

Kruskal's algorithm (see CLRS):

- Uses the ***disjoint-set data structure*** (see CLRS, Ch. 21).
- Running time = $O(E \lg V)$.



MST algorithms

Kruskal's algorithm (see CLRS):

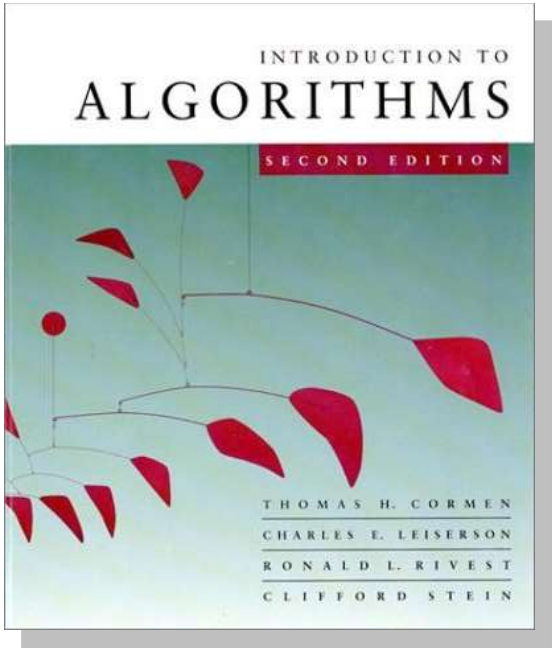
- Uses the ***disjoint-set data structure*** (see CLRS, Ch. 21).
- Running time = $O(E \lg V)$.

Best to date:

- Karger, Klein, and Tarjan [1993].
- Randomized algorithm.
- $O(V + E)$ expected time.

Introduction to Algorithms

6.046J/18.401J

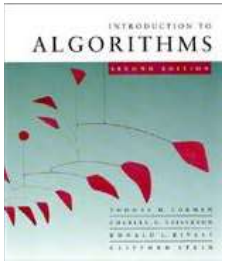


LECTURE 17

Shortest Paths I

- Properties of shortest paths
- Dijkstra's algorithm
- Correctness
- Analysis
- Breadth-first search

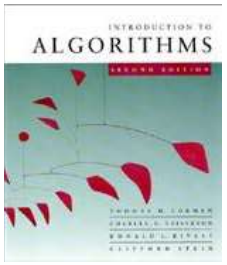
Prof. Erik Demaine



Paths in graphs

Consider a digraph $G = (V, E)$ with edge-weight function $w : E \to \mathsf{R}$. The ***weight*** of path $p = v_1 \to v_2 \to \cdots \to v_k$ is defined to be

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

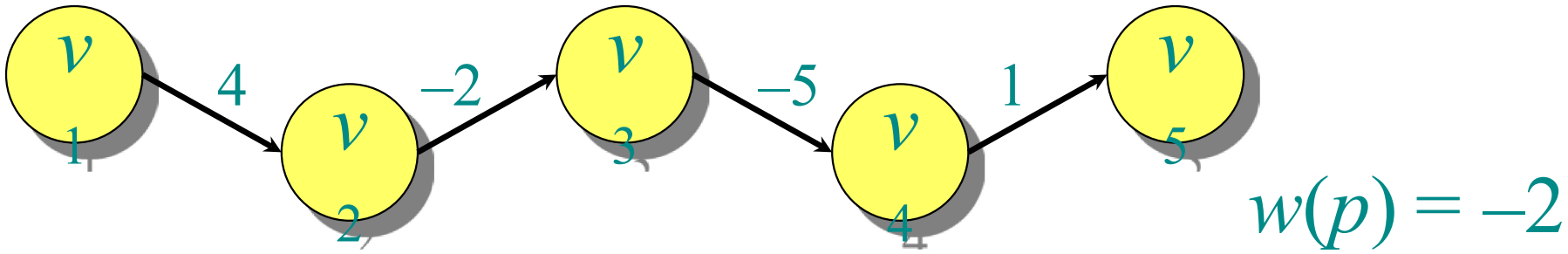


Paths in graphs

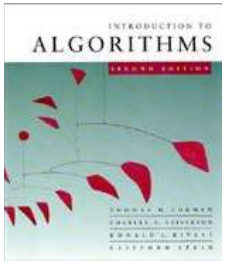
Consider a digraph $G = (V, E)$ with edge-weight function $w : E \to \mathsf{R}$. The ***weight*** of path $p = v_1 \to v_2 \to \cdots \to v_k$ is defined to be

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

Example:



$w(p) = -2$

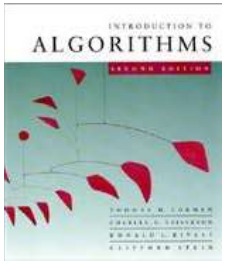


Shortest paths

A ***shortest path*** from u to v is a path of minimum weight from u to v. The ***shortest-path weight*** from u to v is defined as

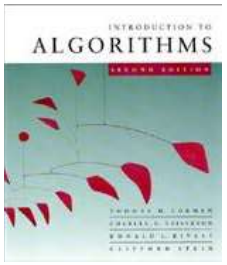
$$\delta(u, v) = \min\{w(p) : p \text{ is a path from } u \text{ to } v\}.$$

Note: $\delta(u, v) = \infty$ if no path from u to v exists.



Well-definedness of shortest paths

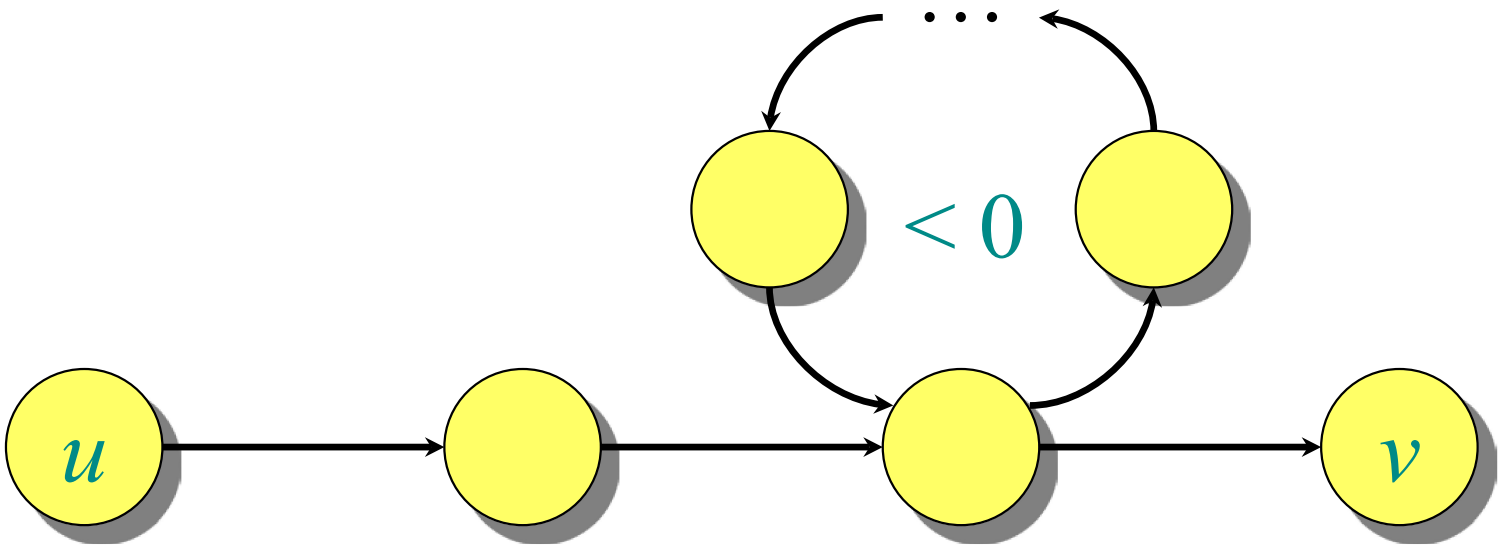
If a graph G contains a negative-weight cycle, then some shortest paths do not exist.

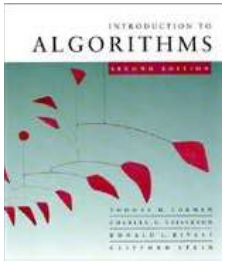


Well-definedness of shortest paths

If a graph G contains a negative-weight cycle, then some shortest paths do not exist.

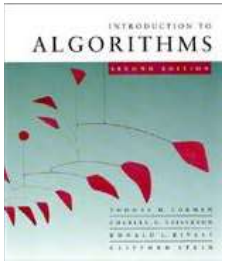
Example:





Optimal substructure

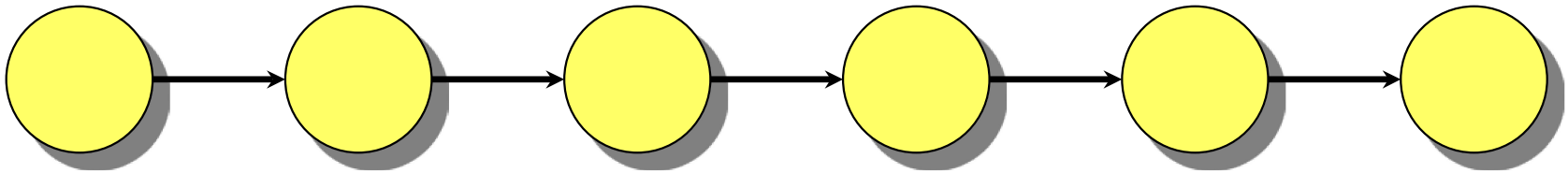
Theorem. A subpath of a shortest path is a shortest path.

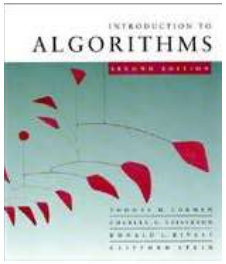


Optimal substructure

Theorem. A subpath of a shortest path is a shortest path.

Proof. Cut and paste:

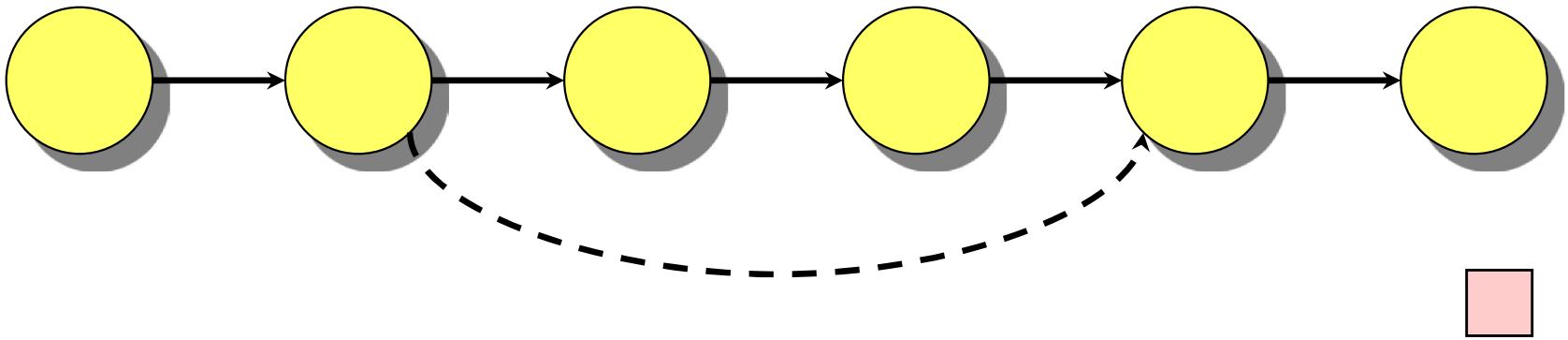


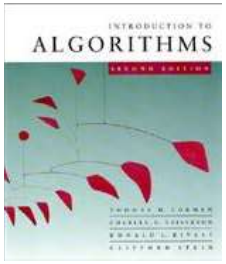


Optimal substructure

Theorem. A subpath of a shortest path is a shortest path.

Proof. Cut and paste:

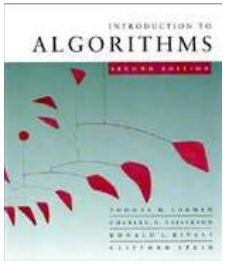




Triangle inequality

Theorem. For all $u, v, x \in V$, we have

$$\delta(u, v) \leq \delta(u, x) + \delta(x, v).$$

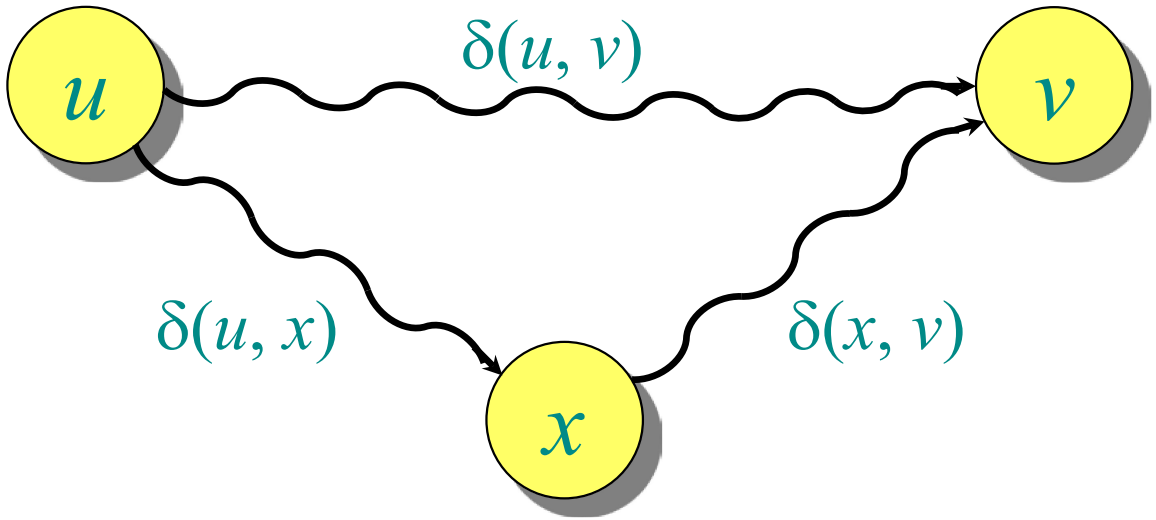


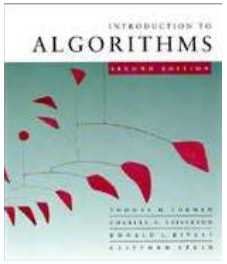
Triangle inequality

Theorem. For all $u, v, x \in V$, we have

$$\delta(u, v) \leq \delta(u, x) + \delta(x, v).$$

Proof.



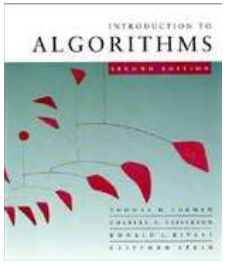


Single-source shortest paths (nonnegative edge weights)

Problem. Assume that $w(u, v) \geq 0$ for all $(u, v) \in E$. (Hence, all shortest-path weights must exist.) From a given source vertex $s \in V$, find the shortest-path weights $\delta(s, v)$ for all $v \in V$.

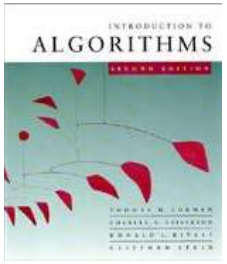
IDEA: Greedy.

1. Maintain a set S of vertices whose shortest-path distances from s are known.
2. At each step, add to S the vertex $v \in V - S$ whose distance estimate from s is minimum.
3. Update the distance estimates of vertices adjacent to v.



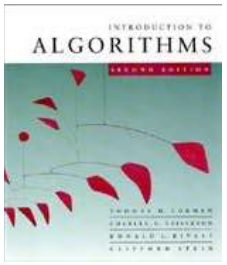
Dijkstra's algorithm

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
  do $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$ ▷ Q is a priority queue maintaining $V - S$, keyed on $d[v]$



Dijkstra's algorithm

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
  do $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$ ▷ Q is a priority queue maintaining $V - S$, keyed on $d[v]$
while $Q \neq \varnothing$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$
    for each $v \in Adj[u]$
      do if $d[v] > d[u] + w(u, v)$
        then $d[v] \leftarrow d[u] + w(u, v)$

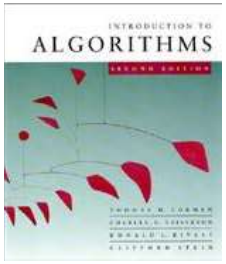


Dijkstra's algorithm

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
  do $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$ ▷ Q is a priority queue maintaining $V - S$, keyed on $d[v]$
while $Q \neq \varnothing$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$
    for each $v \in Adj[u]$
      do if $d[v] > d[u] + w(u, v)$
        then $d[v] \leftarrow d[u] + w(u, v)$

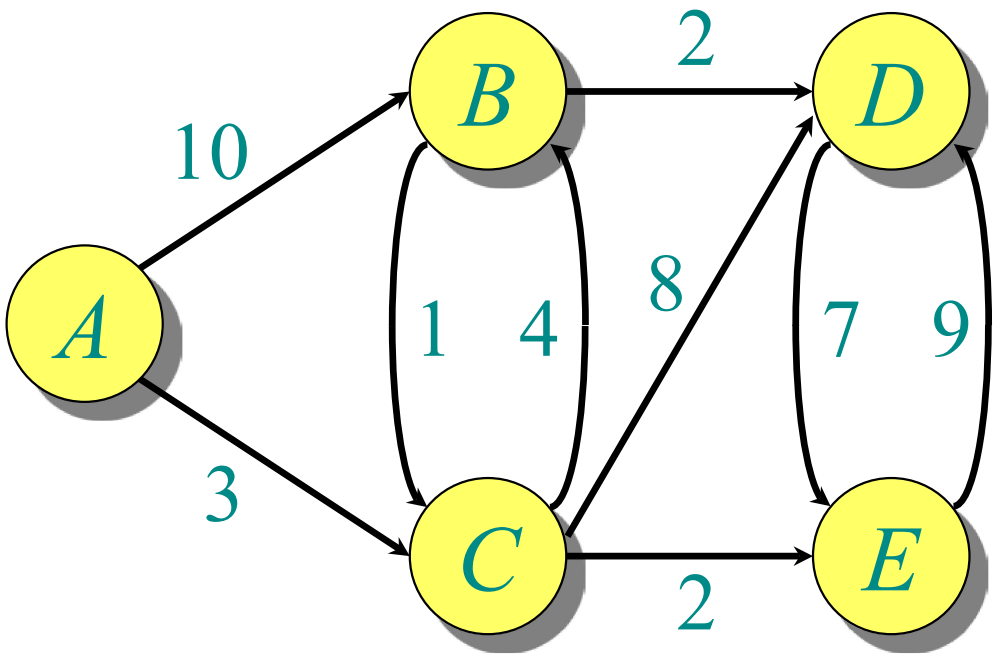
relaxation step

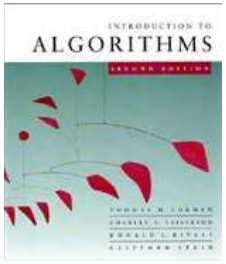
Implicit DECREASE-KEY



Example of Dijkstra's algorithm

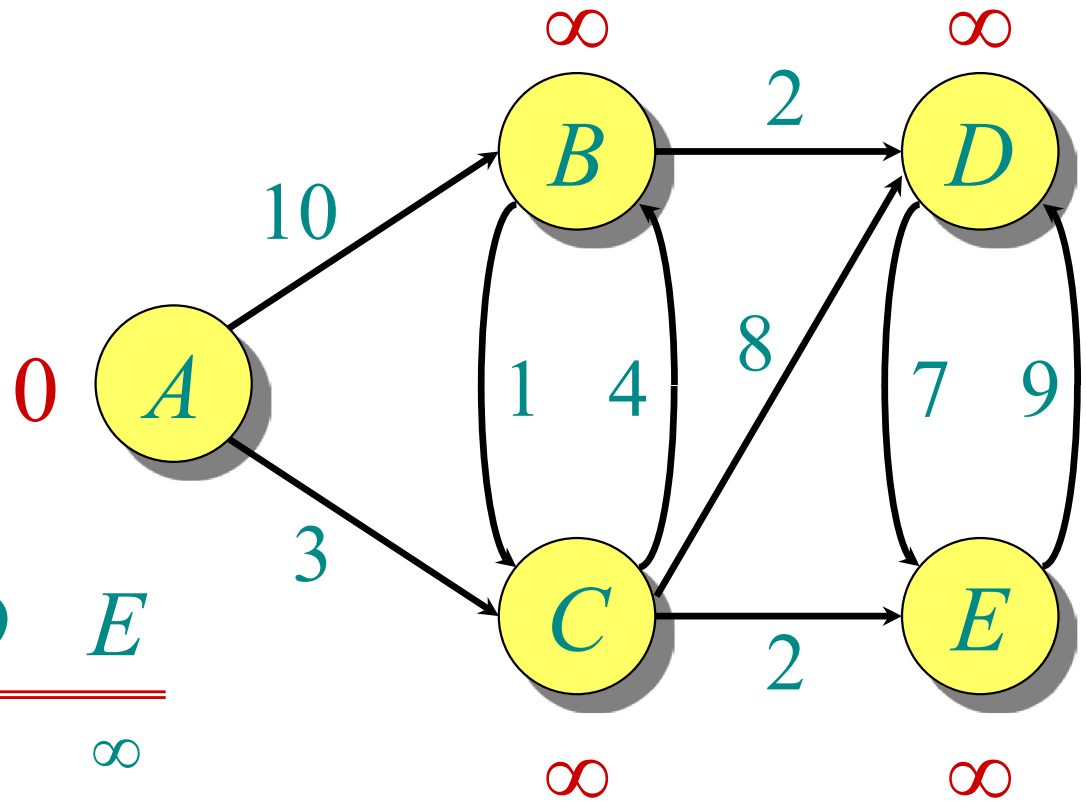
Graph with nonnegative edge weights:





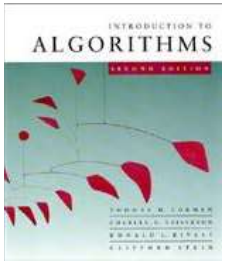
Example of Dijkstra's algorithm

Initialize:



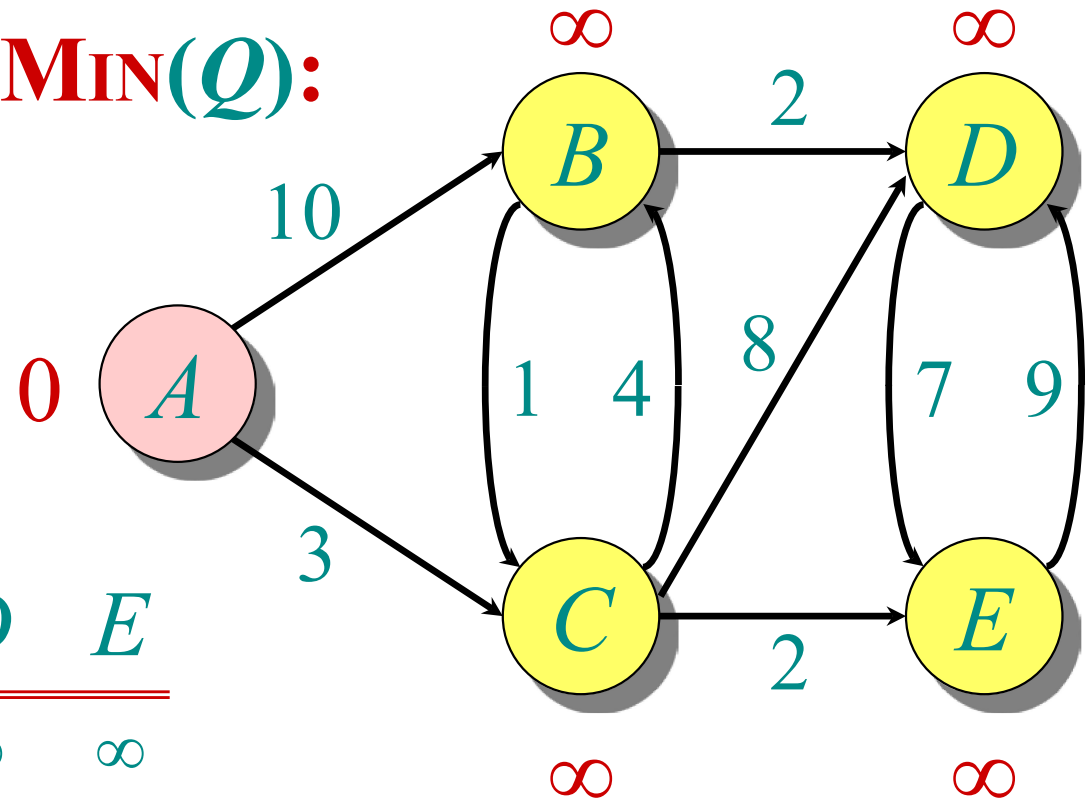
Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: {}



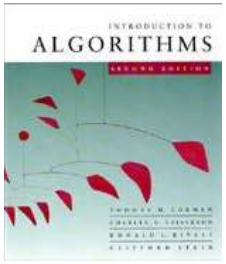
Example of Dijkstra's algorithm

"A" ← EXTRACT-MIN(Q):



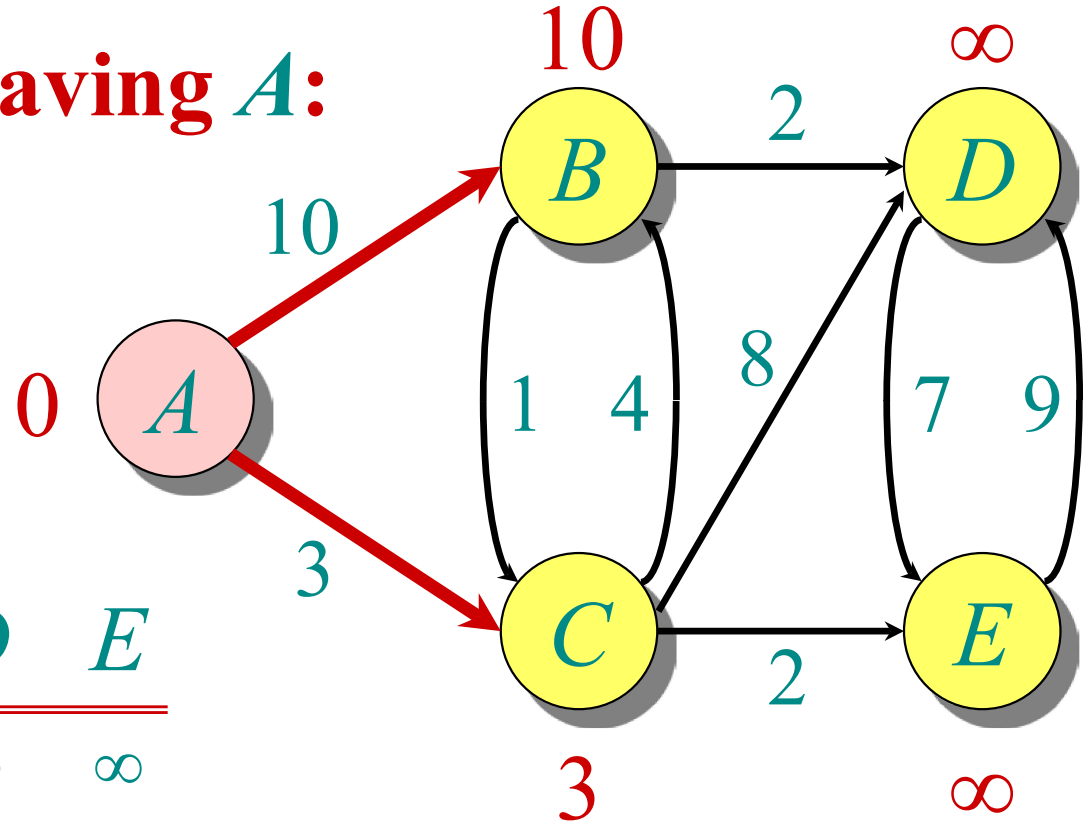
Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: { A }



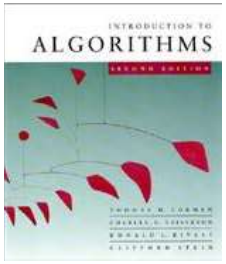
Example of Dijkstra's algorithm

Relax all edges leaving *A*:



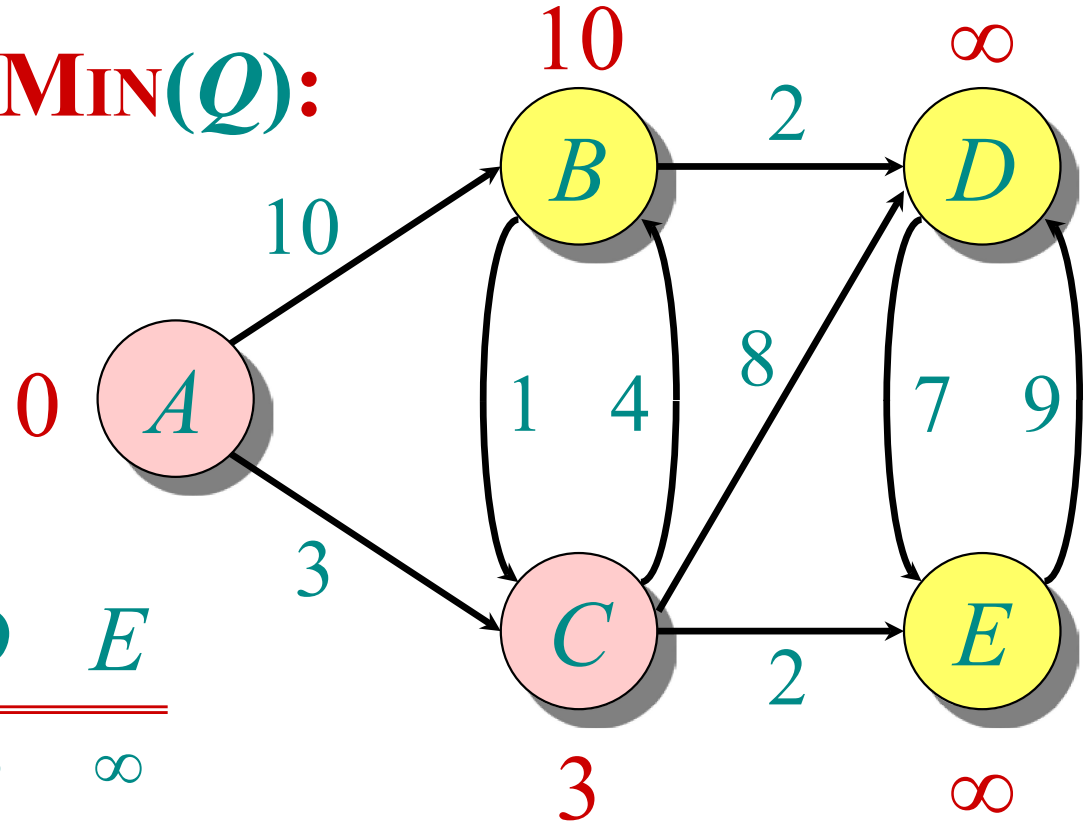
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { *A* }



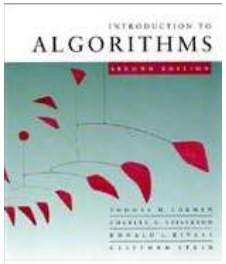
Example of Dijkstra's algorithm

"C" ← EXTRACT-MIN(Q):



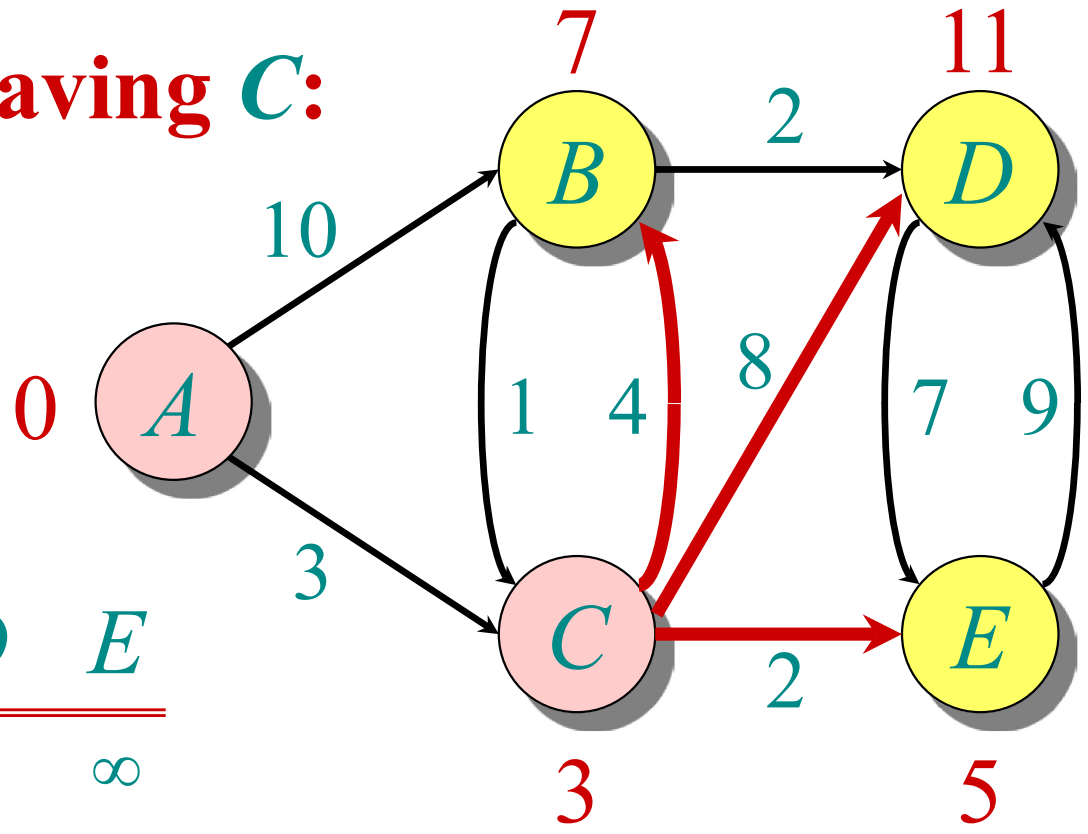
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A, C }



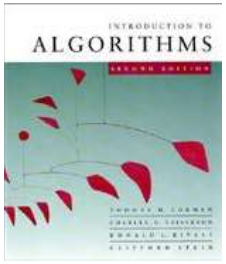
Example of Dijkstra's algorithm

Relax all edges leaving *C*:



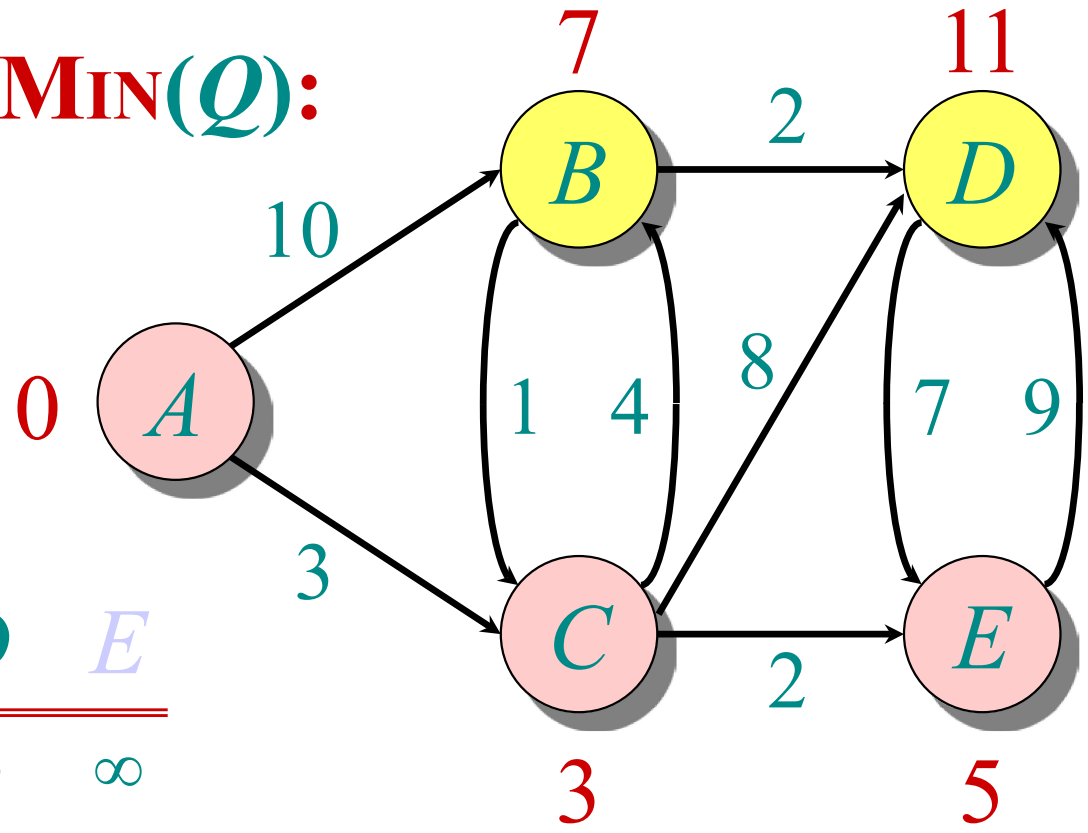
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { *A*, *C* }



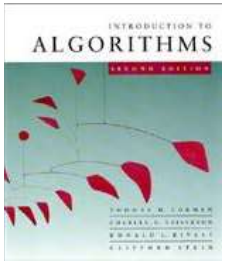
Example of Dijkstra's algorithm

"E" ← EXTRACT-MIN(Q):



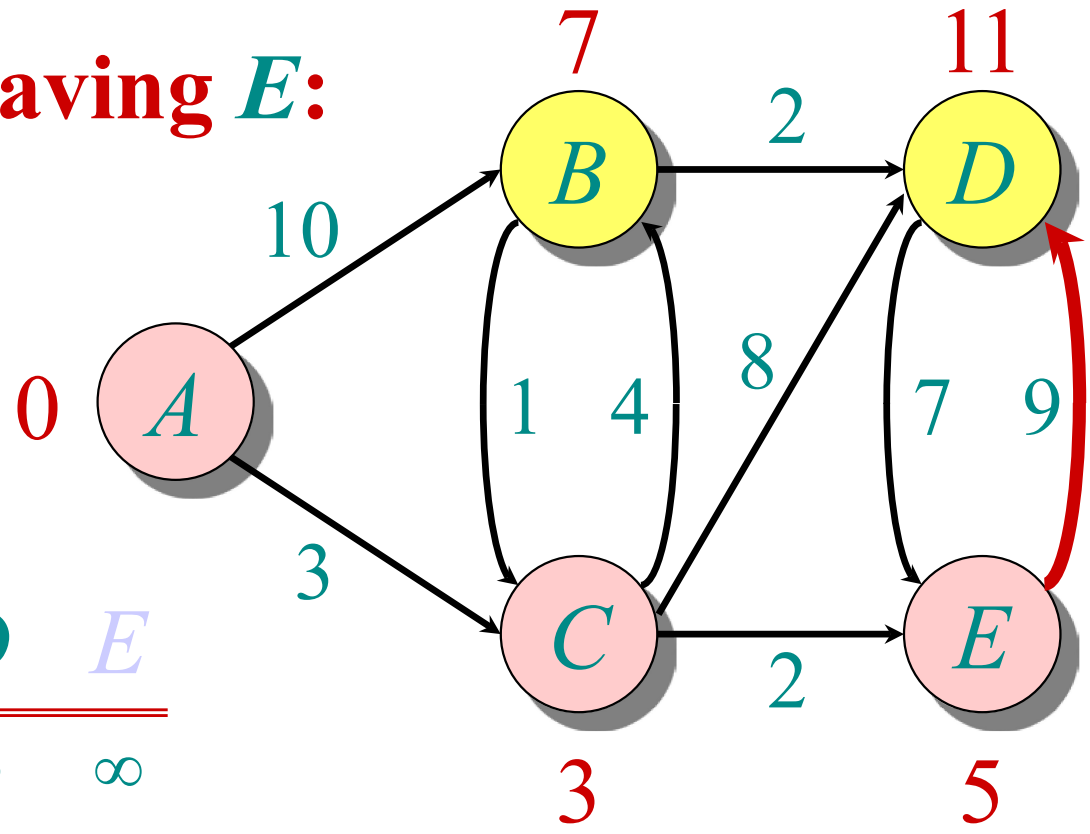
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C, E }



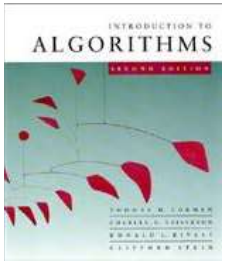
Example of Dijkstra's algorithm

Relax all edges leaving *E*:



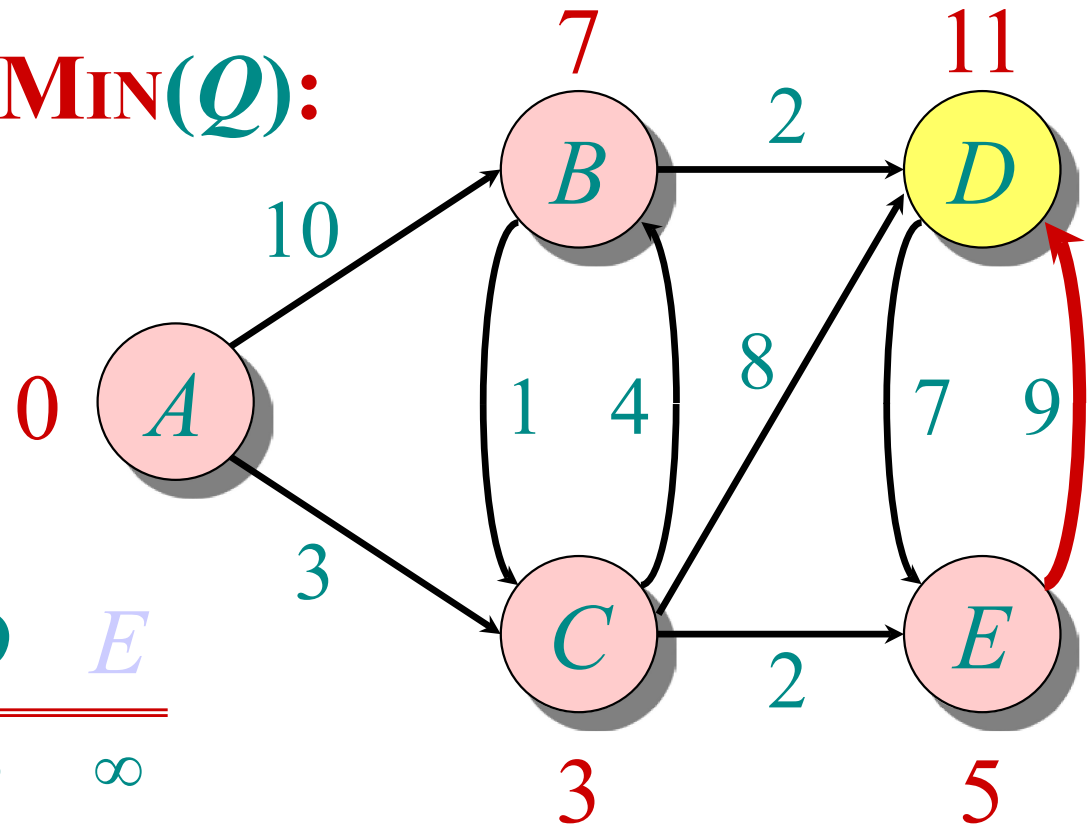
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { *A*, *C*, *E* }



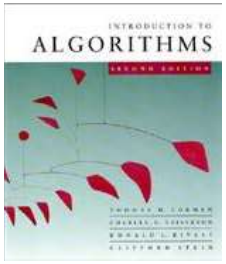
Example of Dijkstra's algorithm

"*B*" ← EXTRACT-MIN(*Q*):



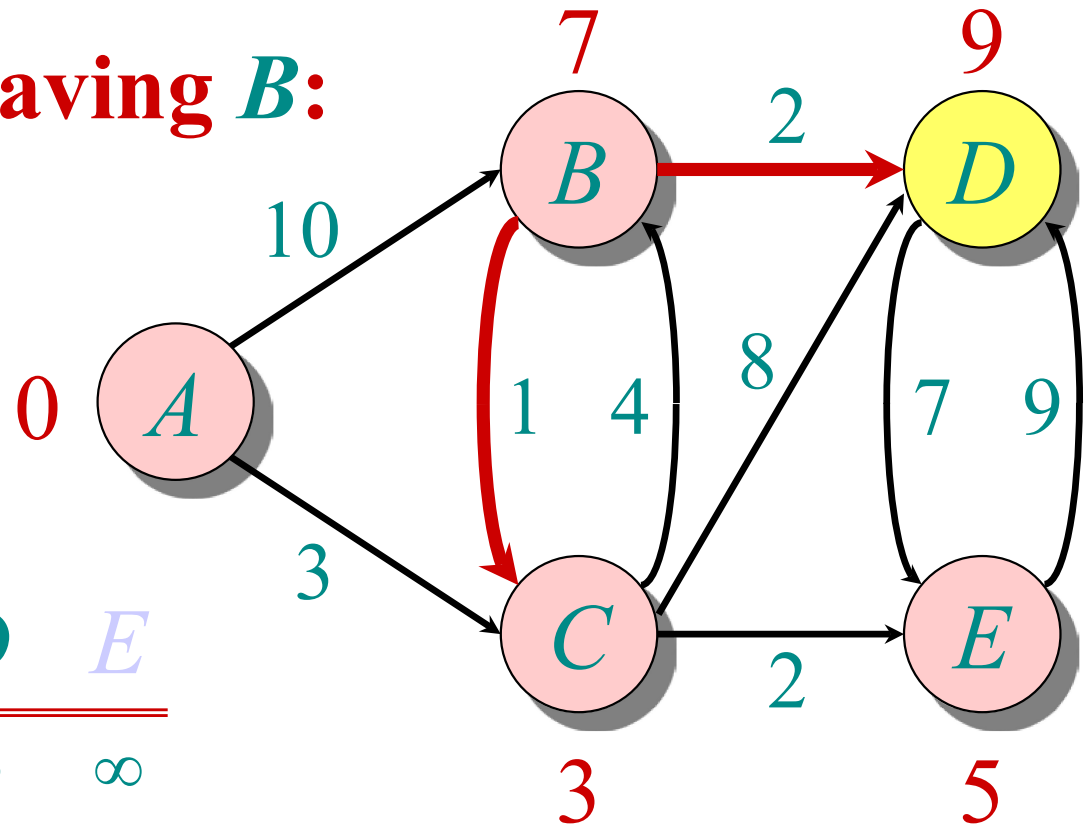
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { *A*, *C*, *E*, *B* }



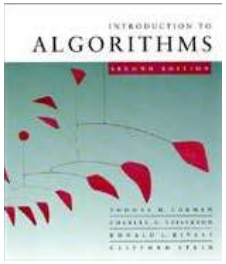
Example of Dijkstra's algorithm

Relax all edges leaving *B*:



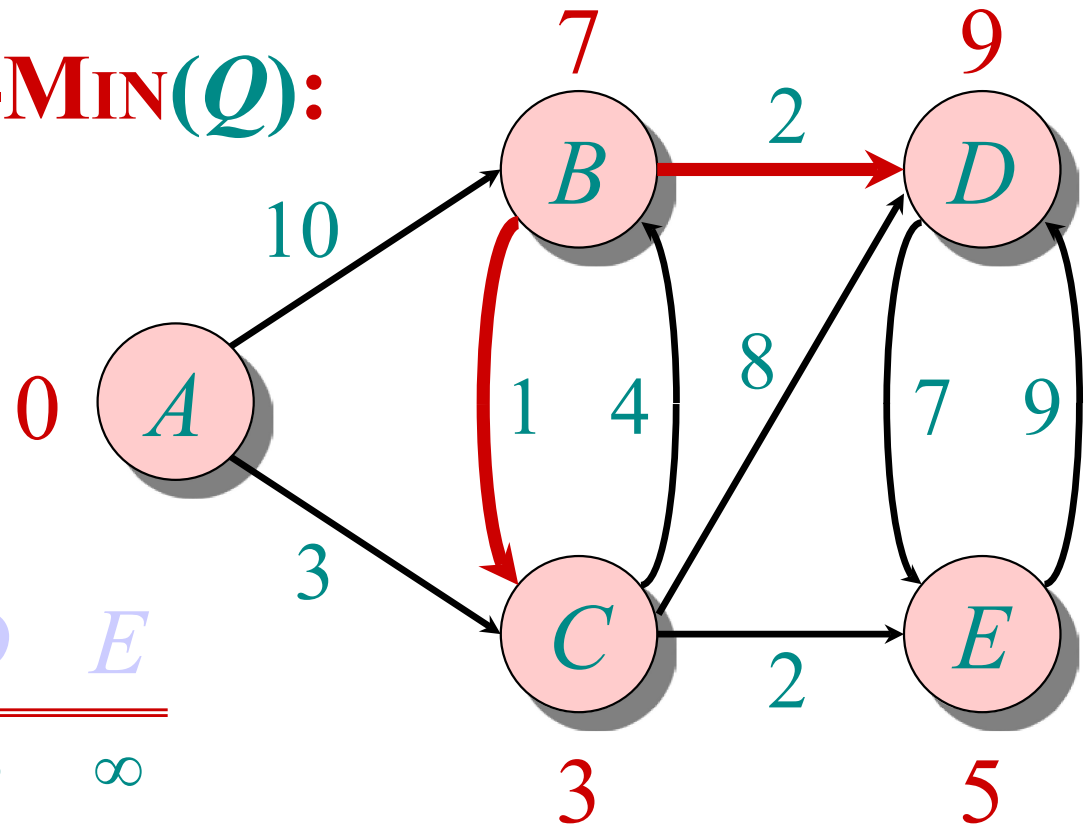
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { *A*, *C*, *E*, *B* }



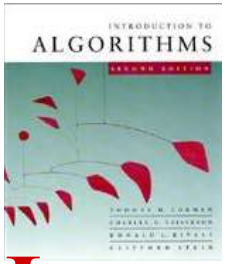
Example of Dijkstra's algorithm

"*D*" ← EXTRACT-MIN(*Q*):



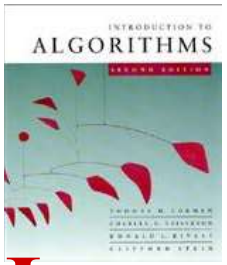
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { *A*, *C*, *E*, *B*, *D* }



Correctness — Part I

Lemma. Initializing $d[s] \leftarrow 0$ and $d[v] \leftarrow \infty$ for all $v \in V - \{s\}$ establishes $d[v] \geq \delta(s, v)$ for all $v \in V$, and this invariant is maintained over any sequence of relaxation steps.



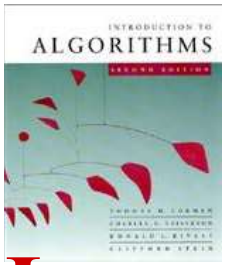
Correctness — Part I

Lemma. Initializing $d[s] \leftarrow 0$ and $d[v] \leftarrow \infty$ for all $v \in V - \{s\}$ establishes $d[v] \geq \delta(s, v)$ for all $v \in V$, and this invariant is maintained over any sequence of relaxation steps.

Proof. Suppose not. Let v be the first vertex for which $d[v] < \delta(s, v)$, and let u be the vertex that caused $d[v]$ to change: $d[v] = d[u] + w(u, v)$. Then,

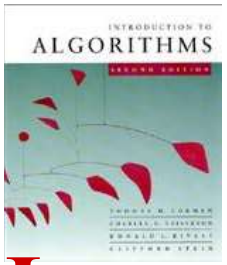
$$\begin{aligned} d[v] &< \delta(s, v) && \text{supposition} \\ &\leq \delta(s, u) + \delta(u, v) && \text{triangle inequality} \\ &\leq \delta(s,u) + w(u, v) && \text{sh. path} \leq \text{specific path} \\ &\leq d[u] + w(u, v) && v \text{ is first violation} \end{aligned}$$

Contradiction. ∎



Correctness — Part II

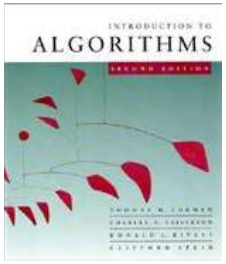
Lemma. Let u be v's predecessor on a shortest path from s to v. Then, if $d[u] = \delta(s, u)$ and edge (u, v) is relaxed, we have $d[v] = \delta(s, v)$ after the relaxation.



Correctness — Part II

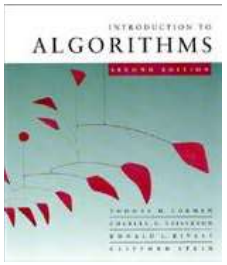
Lemma. Let u be v's predecessor on a shortest path from s to v. Then, if $d[u] = \delta(s, u)$ and edge (u, v) is relaxed, we have $d[v] = \delta(s, v)$ after the relaxation.

Proof. Observe that $\delta(s, v) = \delta(s, u) + w(u, v)$. Suppose that $d[v] > \delta(s, v)$ before the relaxation. (Otherwise, we're done.) Then, the test $d[v] > d[u] + w(u, v)$ succeeds, because $d[v] > \delta(s, v) = \delta(s, u) + w(u, v) = d[u] + w(u, v)$, and the algorithm sets $d[v] = d[u] + w(u, v) = \delta(s, v)$. □



Correctness — Part III

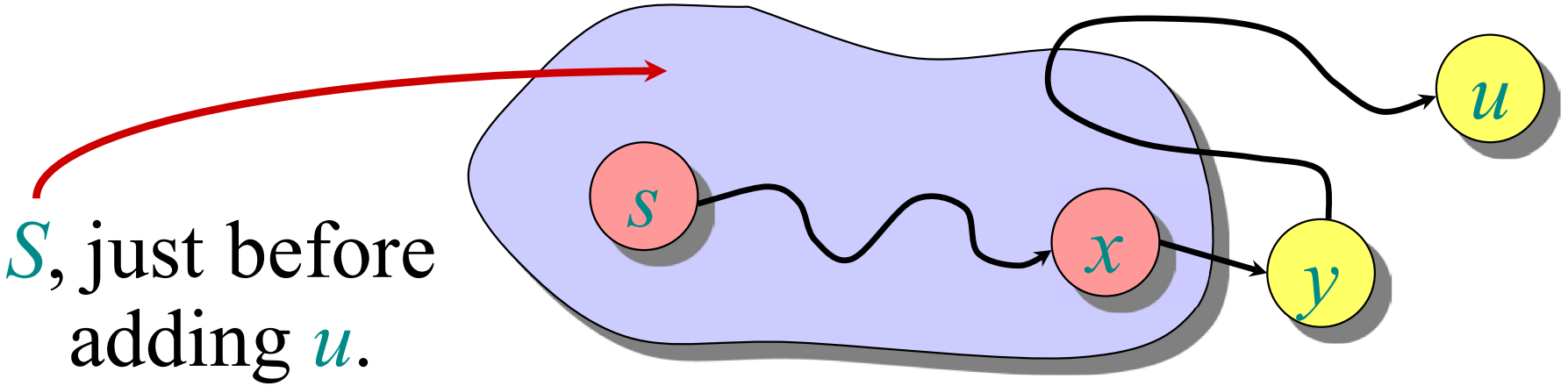
Theorem. Dijkstra's algorithm terminates with $d[v] = \delta(s, v)$ for all $v \in V$.

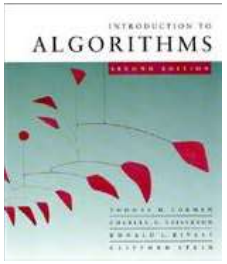


Correctness — Part III

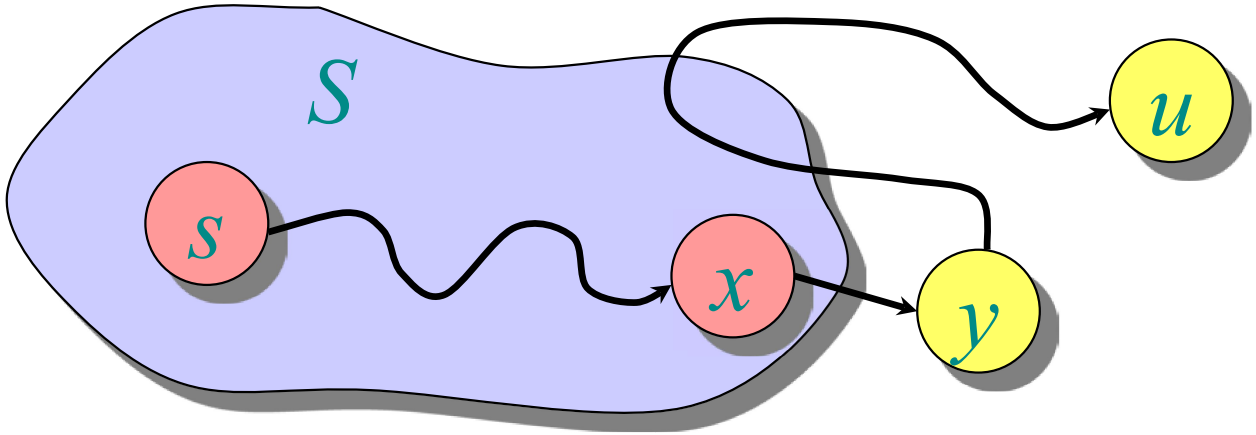
Theorem. Dijkstra's algorithm terminates with $d[v] = \delta(s, v)$ for all $v \in V$.

Proof. It suffices to show that $d[v] = \delta(s, v)$ for every $v \in V$ when v is added to S. Suppose u is the first vertex added to S for which $d[u] > \delta(s, u)$. Let y be the first vertex in $V - S$ along a shortest path from s to u, and let x be its predecessor:

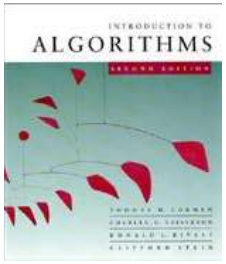




Correctness — Part III (continued)

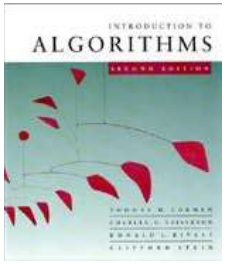


Since u is the first vertex violating the claimed invariant, we have $d[x] = \delta(s, x)$. When x was added to S, the edge (x, y) was relaxed, which implies that $d[y] = \delta(s, y) \leq \delta(s, u) < d[u]$. But, $d[u] \leq d[y]$ by our choice of u. Contradiction. □



Analysis of Dijkstra

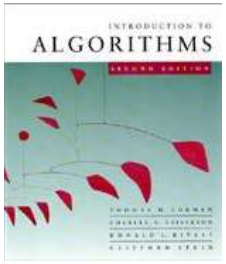
while $Q \neq \varnothing$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$
    for each $v \in Adj[u]$
      do if $d[v] > d[u] + w(u, v)$
        then $d[v] \leftarrow d[u] + w(u, v)$



Analysis of Dijkstra

$|V|$ times

while $Q \neq \varnothing$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$
    for each $v \in Adj[u]$
      do if $d[v] > d[u] + w(u, v)$
        then $d[v] \leftarrow d[u] + w(u, v)$



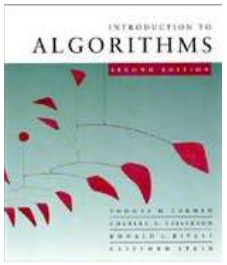
Analysis of Dijkstra

$|V|$ times

while $Q \neq \varnothing$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$

$degree(u)$ times

for each $v \in Adj[u]$
  do if $d[v] > d[u] + w(u, v)$
    then $d[v] \leftarrow d[u] + w(u, v)$

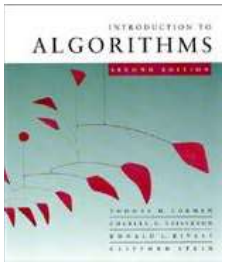


Analysis of Dijkstra

$|V|$ times

while $Q \neq \varnothing$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$
    $degree(u)$ times
    for each $v \in Adj[u]$
      do if $d[v] > d[u] + w(u, v)$
        then $d[v] \leftarrow d[u] + w(u, v)$

Handshaking Lemma $\Rightarrow \Theta(E)$ implicit DECREASE-KEY's.



Analysis of Dijkstra

$|V|$ times

while $Q \neq \varnothing$
do $u \leftarrow$ EXTRACT-MIN(Q)
$S \leftarrow S \cup \{u\}$

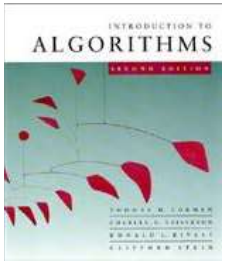
$degree(u)$ times

for each $v \in Adj[u]$
do if $d[v] > d[u] + w(u, v)$
then $d[v] \leftarrow d[u] + w(u, v)$

Handshaking Lemma $\Rightarrow \Theta(E)$ implicit DECREASE-KEY's.

Time $= \Theta(V \cdot T_{\text{EXTRACT-MIN}} + E \cdot T_{\text{DECREASE-KEY}})$

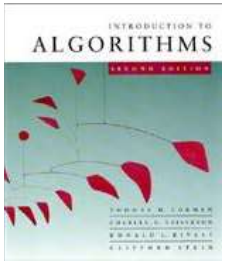
Note: Same formula as in the analysis of Prim's minimum spanning tree algorithm.



Analysis of Dijkstra (continued)

$$\text{Time} = \Theta(V)\cdot T_{\text{EXTRACT-MIN}} + \Theta(E)\cdot T_{\text{DECREASE-KEY}}$$

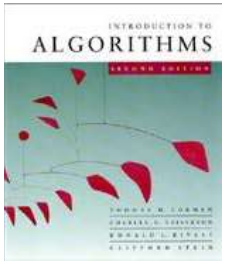
Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total



Analysis of Dijkstra (continued)

$$\text{Time} = \Theta(V)\cdot T_{\text{EXTRACT-MIN}} + \Theta(E)\cdot T_{\text{DECREASE-KEY}}$$

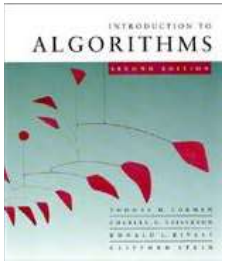
Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$



Analysis of Dijkstra (continued)

$$\text{Time} = \Theta(V)\cdot T_{\text{EXTRACT-MIN}} + \Theta(E)\cdot T_{\text{DECREASE-KEY}}$$

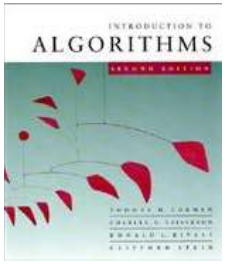
Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$



Analysis of Dijkstra (continued)

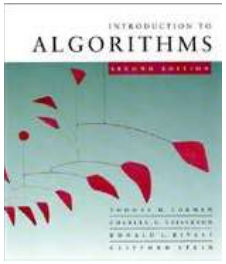
$$\text{Time} = \Theta(V)\cdot T_{\text{EXTRACT-MIN}} + \Theta(E)\cdot T_{\text{DECREASE-KEY}}$$

Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap	$O(\lg V)$ amortized	$O(1)$ amortized	$O(E + V \lg V)$ worst case



Unweighted graphs

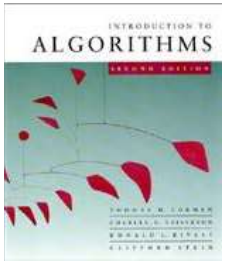
Suppose that $w(u, v) = 1$ for all $(u, v) \in E$.
Can Dijkstra's algorithm be improved?



Unweighted graphs

Suppose that $w(u, v) = 1$ for all $(u, v) \in E$.
Can Dijkstra's algorithm be improved?

- Use a simple FIFO queue instead of a priority queue.



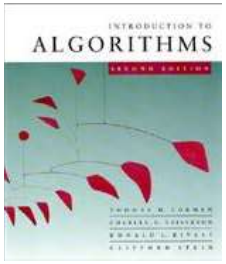
Unweighted graphs

Suppose that $w(u, v) = 1$ for all $(u, v) \in E$.
Can Dijkstra's algorithm be improved?

- Use a simple FIFO queue instead of a priority queue.

Breadth-first search

while $Q \neq \varnothing$
  do $u \leftarrow$ DEQUEUE(Q)
    for each $v \in Adj[u]$
      do if $d[v] = \infty$
        then $d[v] \leftarrow d[u] + 1$
          ENQUEUE(Q, v)



Unweighted graphs

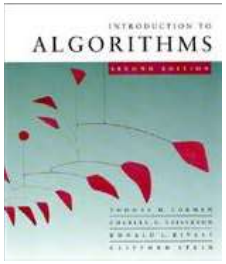
Suppose that $w(u, v) = 1$ for all $(u, v) \in E$.
Can Dijkstra's algorithm be improved?

- Use a simple FIFO queue instead of a priority queue.

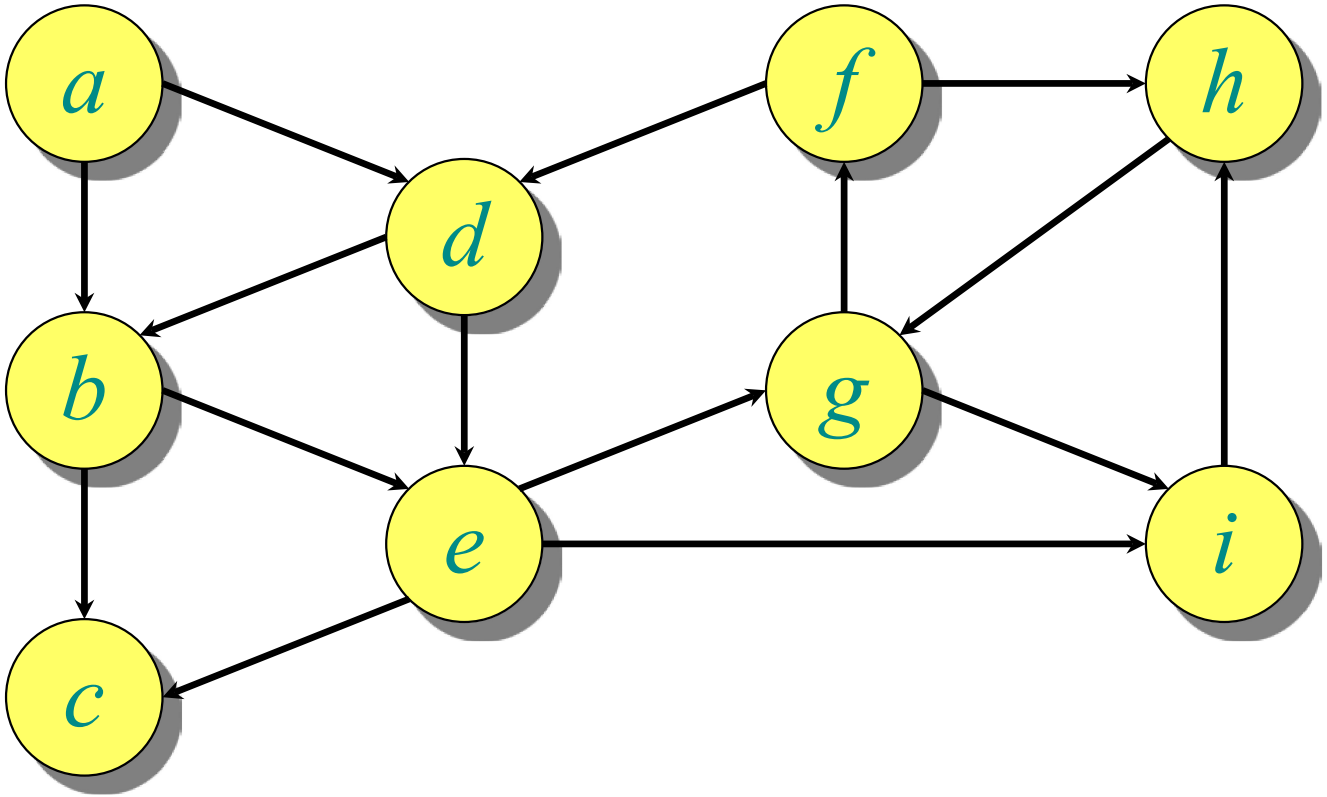
Breadth-first search

while $Q \neq \varnothing$
  do $u \leftarrow$ DEQUEUE(Q)
    for each $v \in Adj[u]$
      do if $d[v] = \infty$
        then $d[v] \leftarrow d[u] + 1$
          ENQUEUE(Q, v)

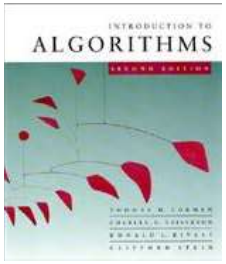
Analysis: Time $= O(V + E)$.



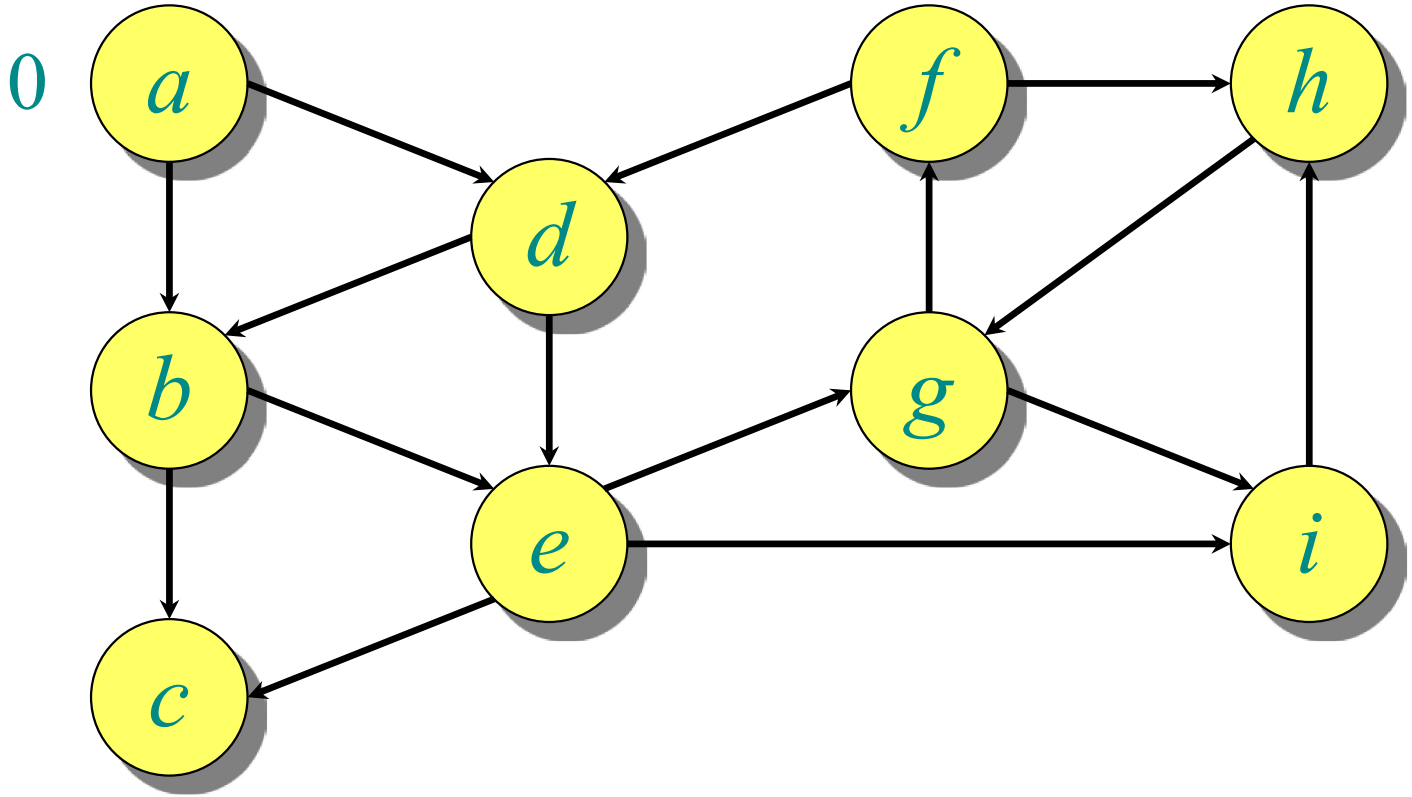
Example of breadth-first search



Q:

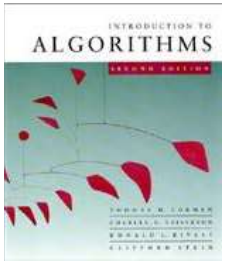


Example of breadth-first search

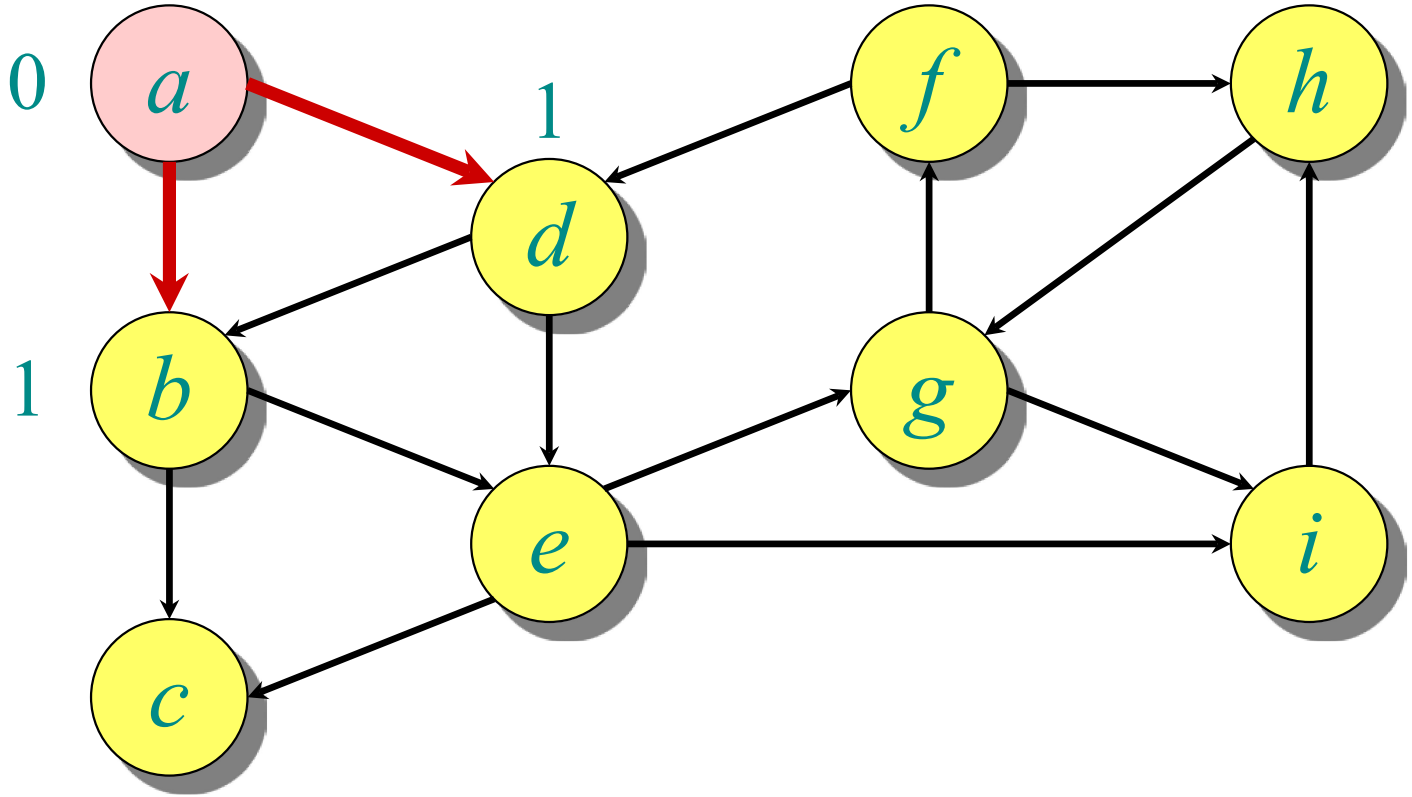


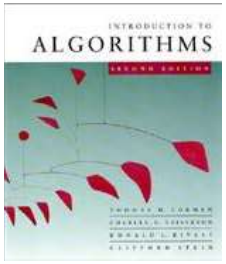
0

Q: a

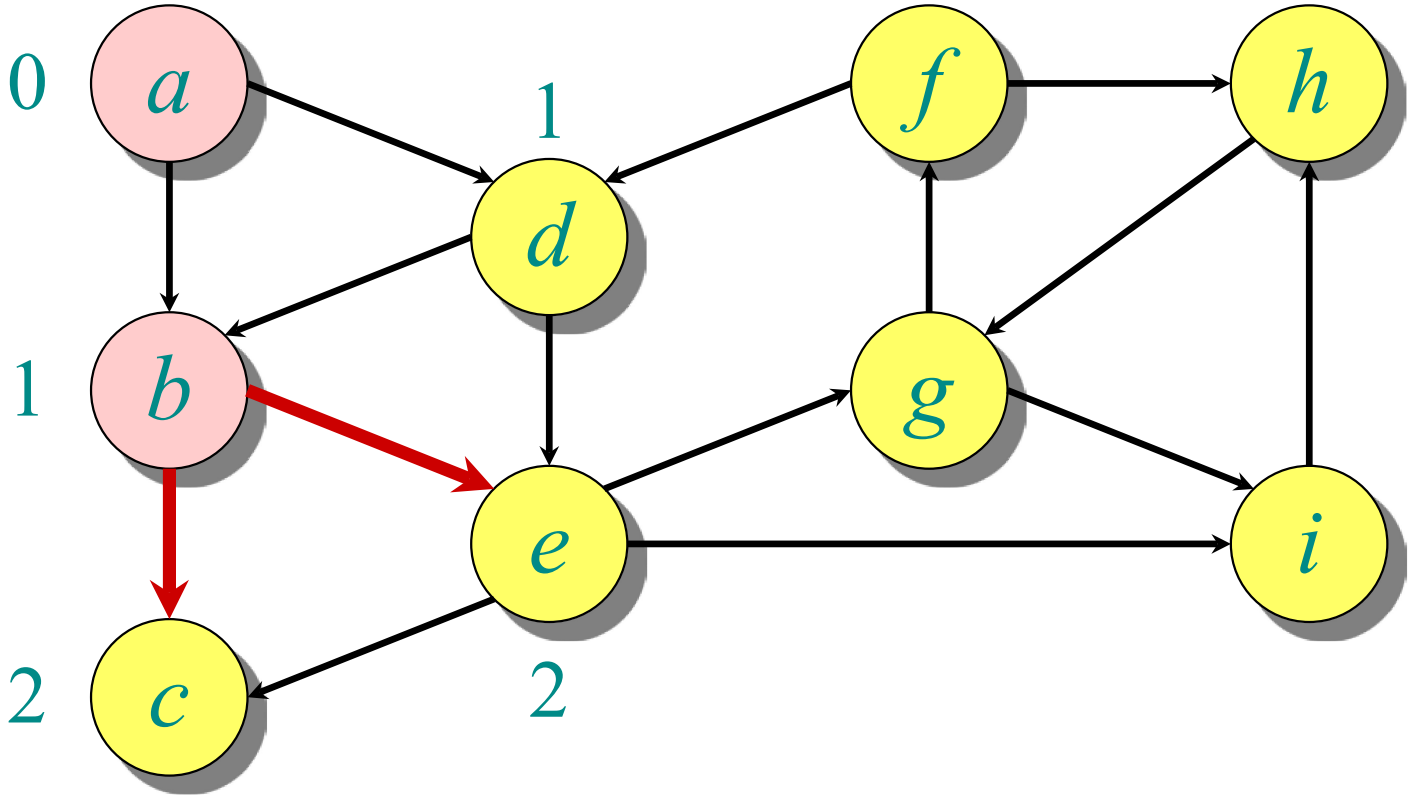


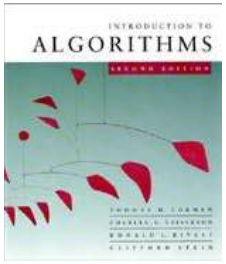
Example of breadth-first search



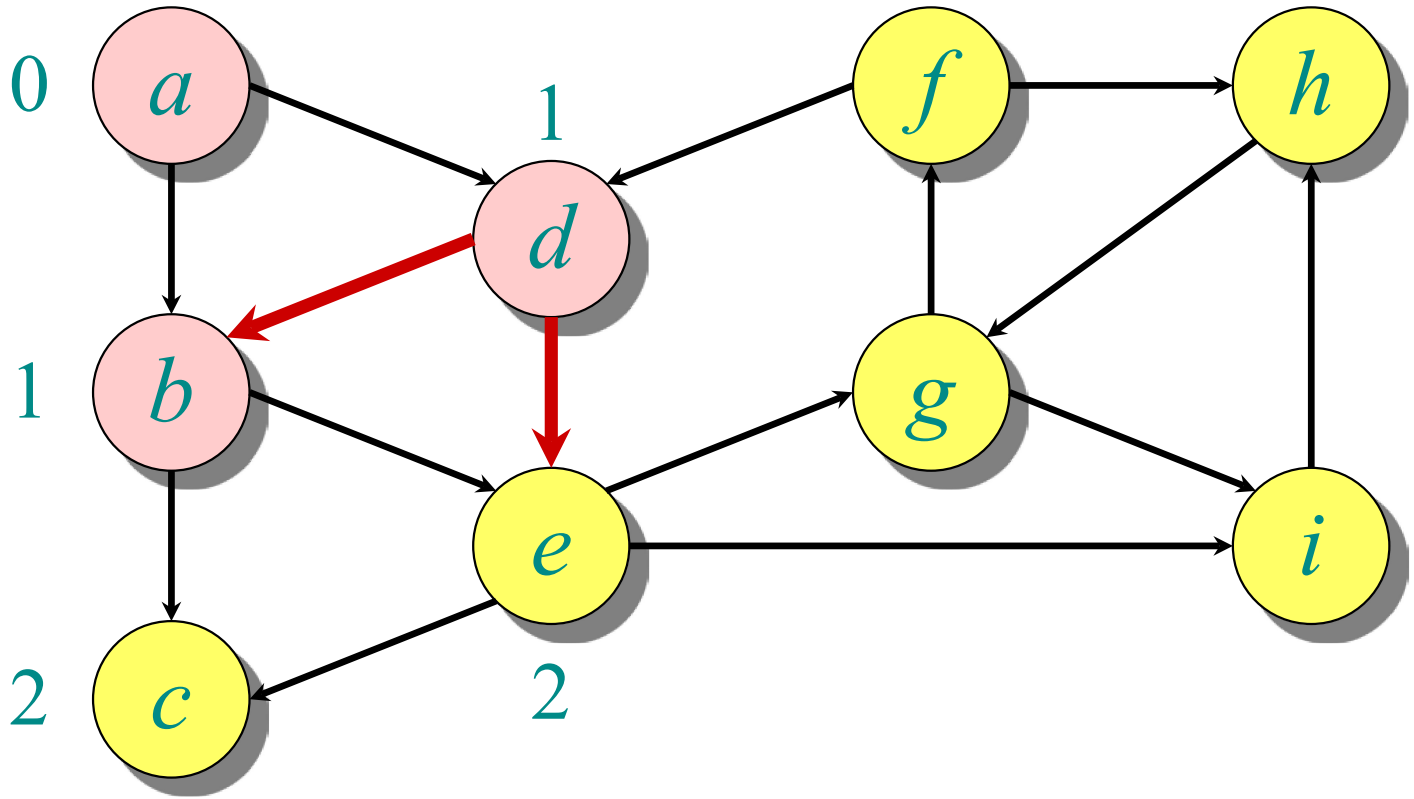


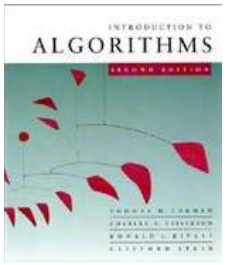
Example of breadth-first search



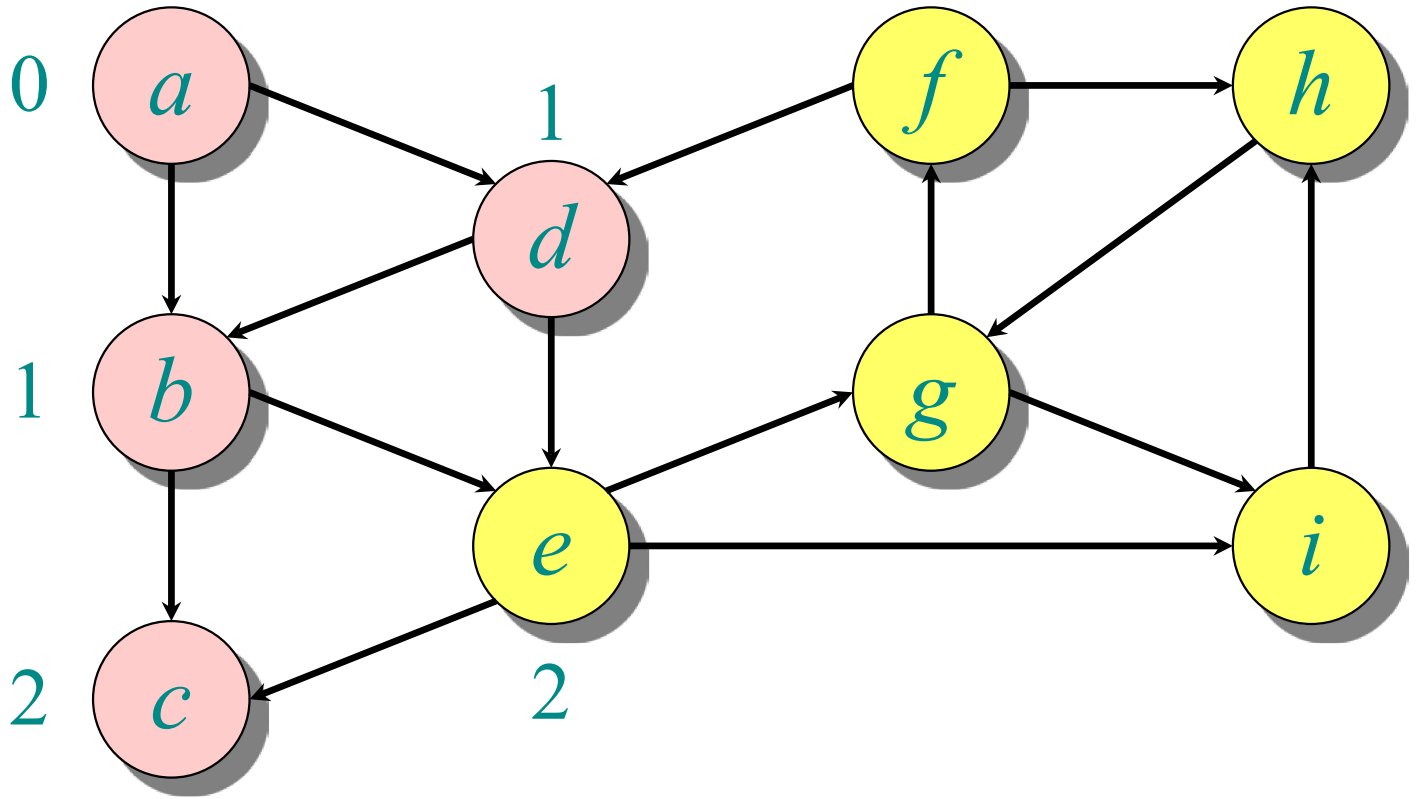


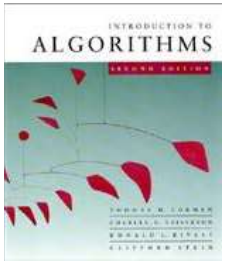
Example of breadth-first search



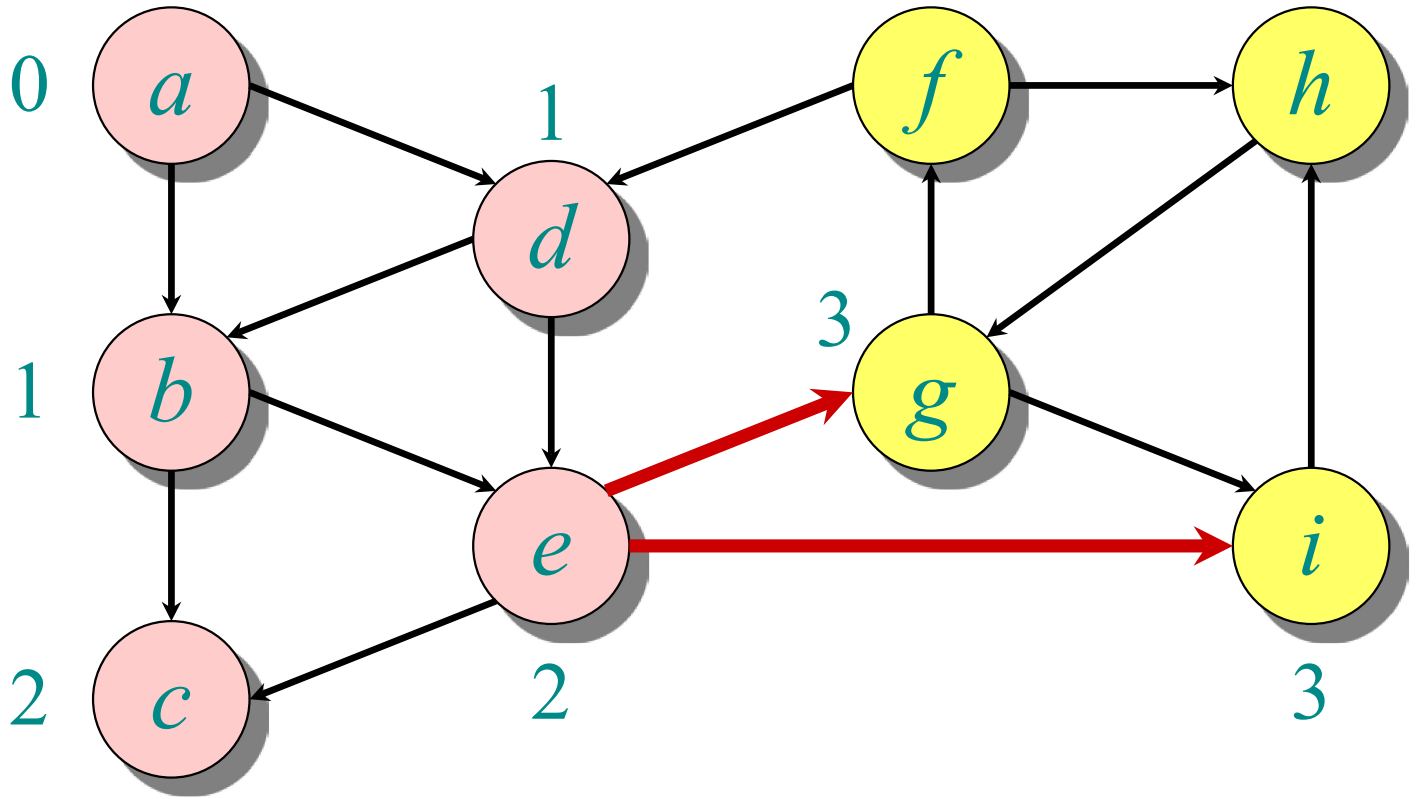


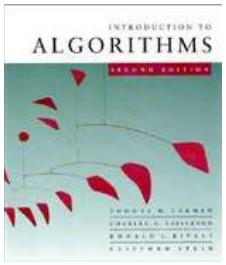
Example of breadth-first search



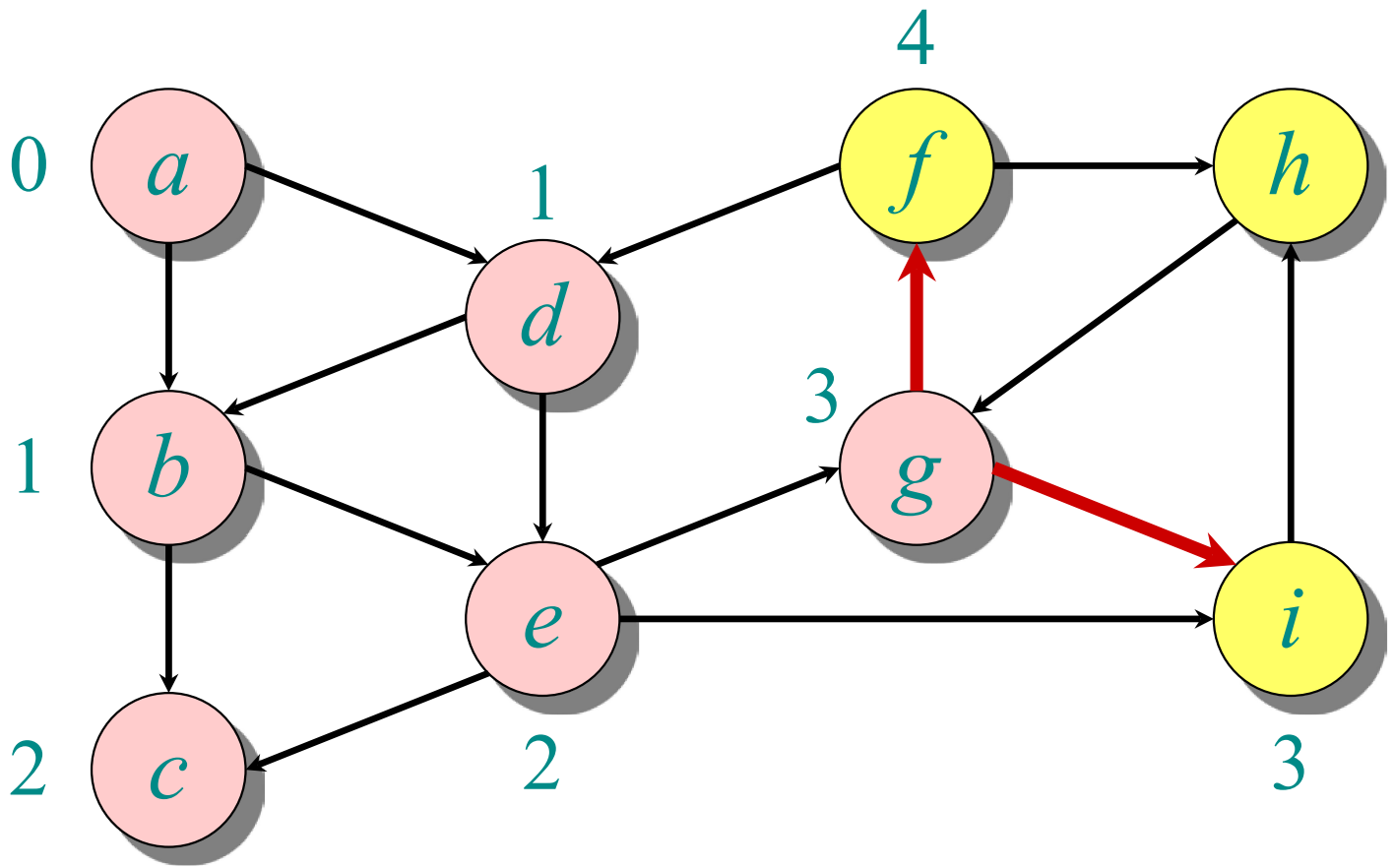


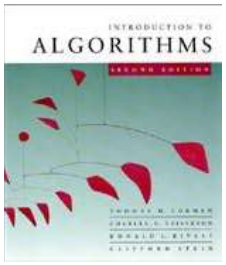
Example of breadth-first search



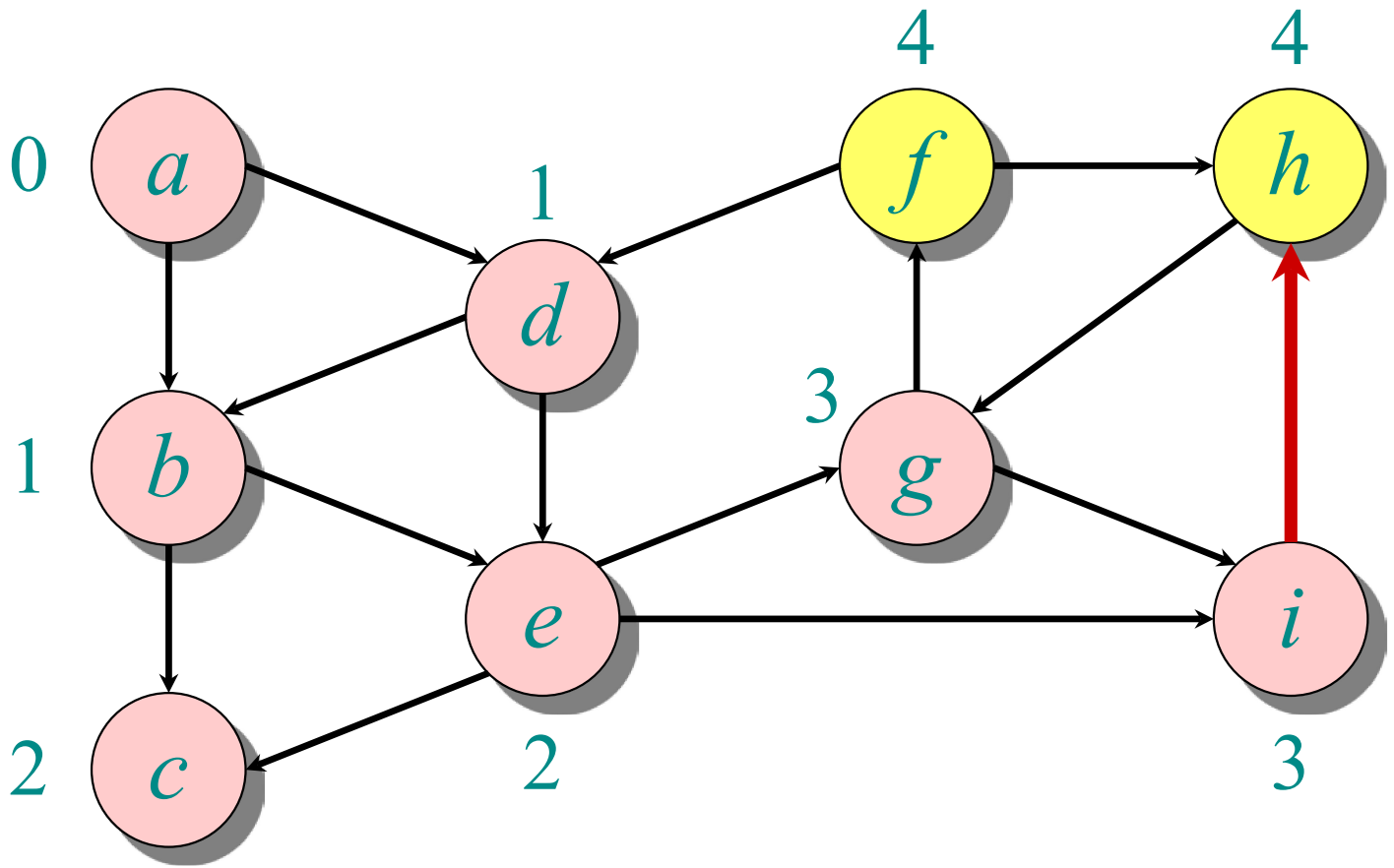


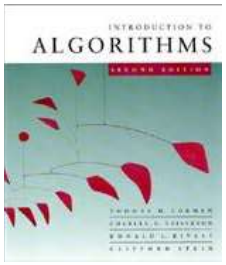
Example of breadth-first search



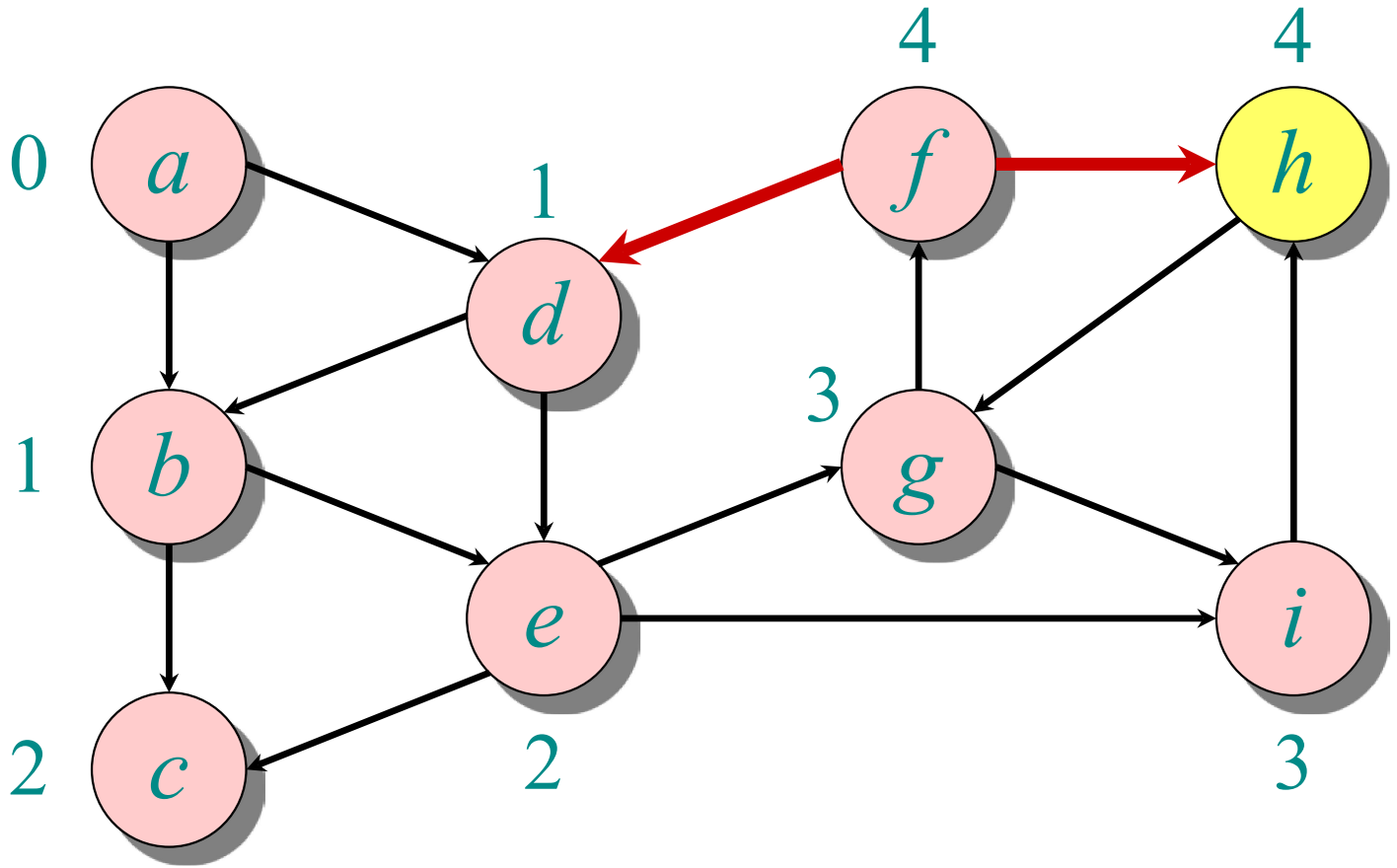


Example of breadth-first search

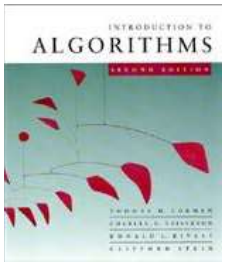




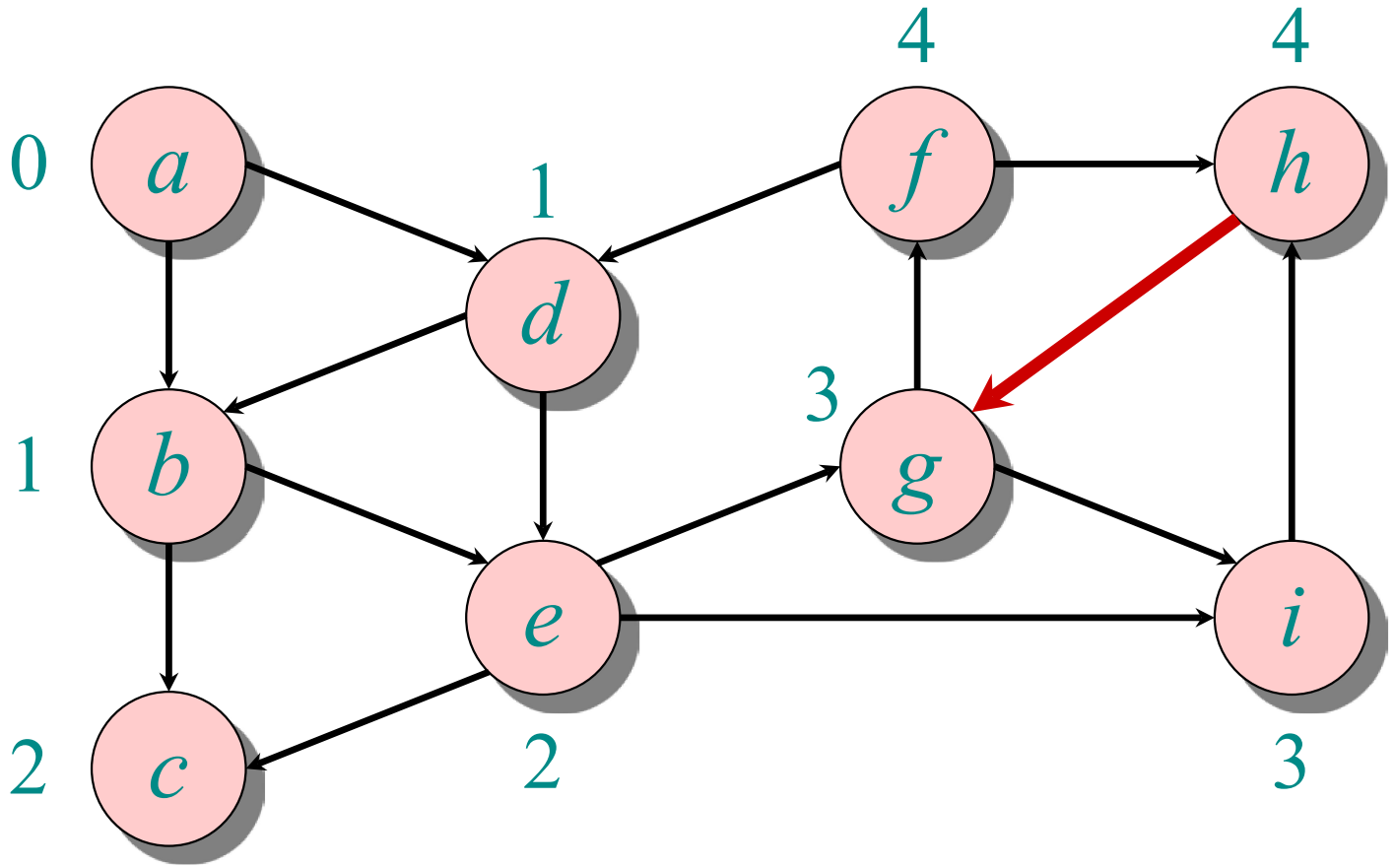
Example of breadth-first search



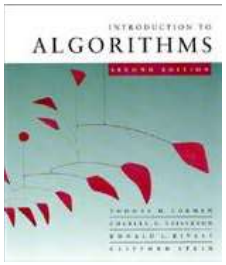
Q: a b d c e g i f h



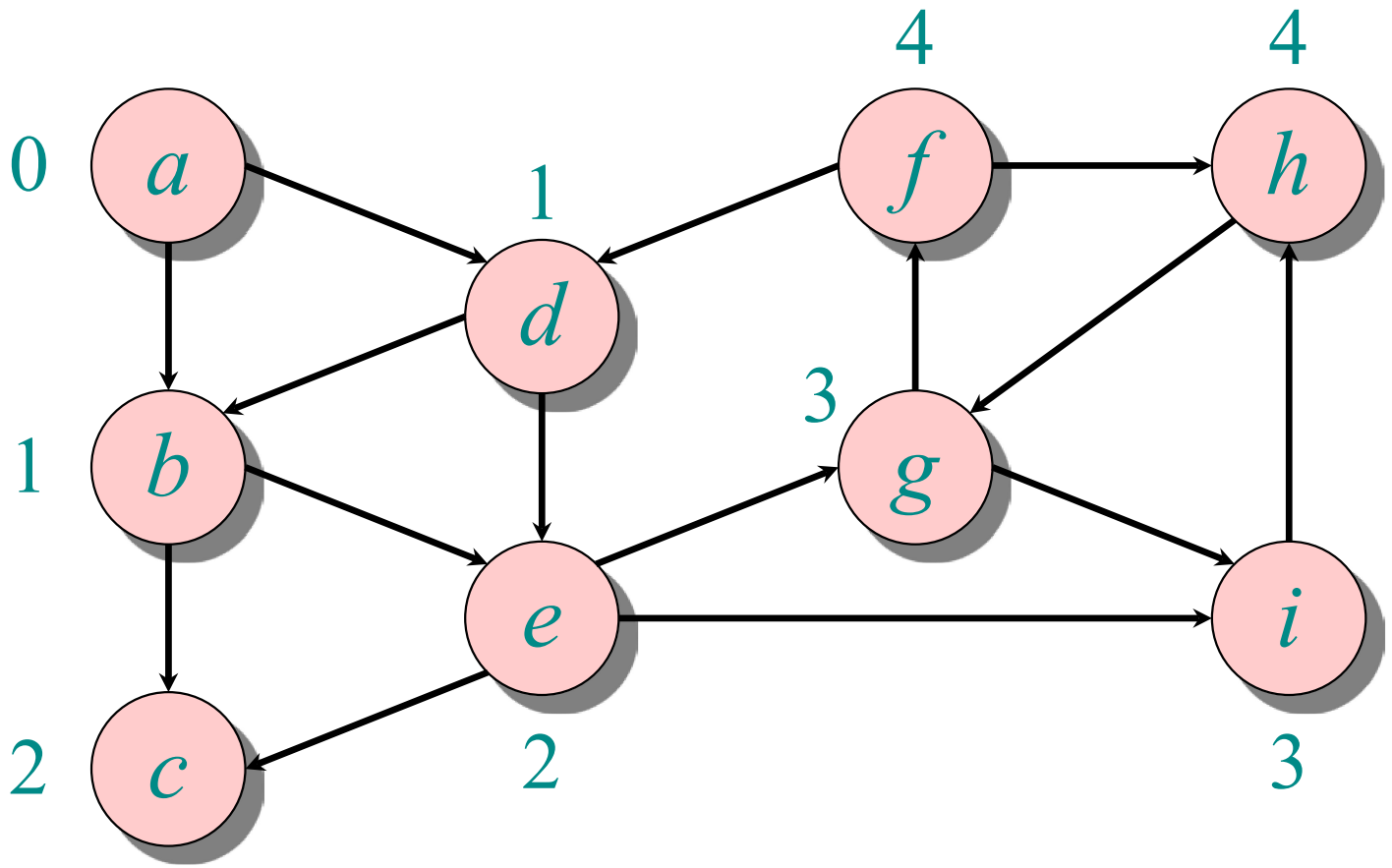
Example of breadth-first search



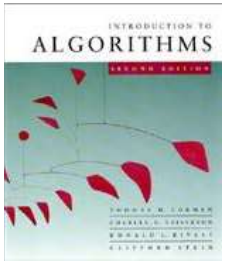
Q: a b d c e g i f h



Example of breadth-first search



Q: a b d c e g i f h



Correctness of BFS

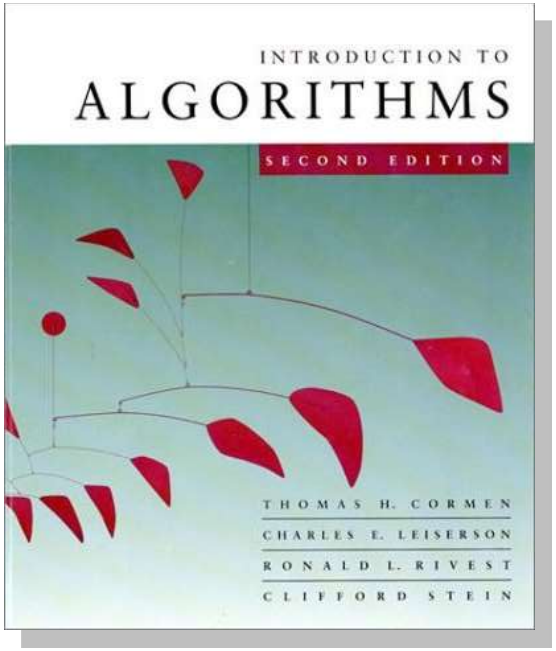
while $Q \neq \varnothing$
  do $u \leftarrow$ DEQUEUE(Q)
    for each $v \in Adj[u]$
      do if $d[v] = \infty$
        then $d[v] \leftarrow d[u] + 1$
          ENQUEUE(Q, v)

Key idea:

The FIFO Q in breadth-first search mimics the priority queue Q in Dijkstra.

- **Invariant:** v comes after u in Q implies that $d[v] = d[u]$ or $d[v] = d[u] + 1$.

Introduction to Algorithms
6.046J/18.401J

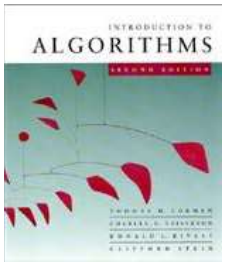


LECTURE 18

Shortest Paths II

- Bellman-Ford algorithm
- Linear programming and difference constraints
- VLSI layout compaction

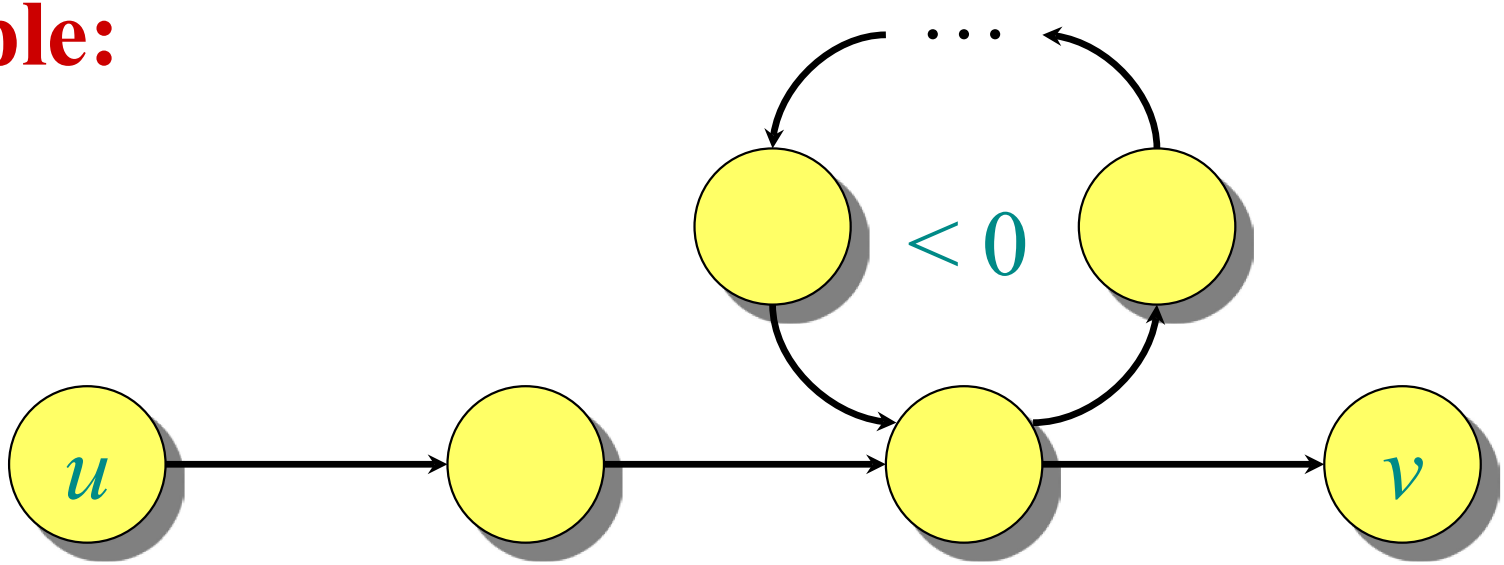
Prof. Erik Demaine

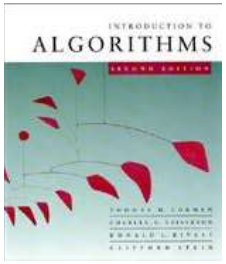


Negative-weight cycles

Recall: If a graph $G = (V, E)$ contains a negative-weight cycle, then some shortest paths may not exist.

Example:

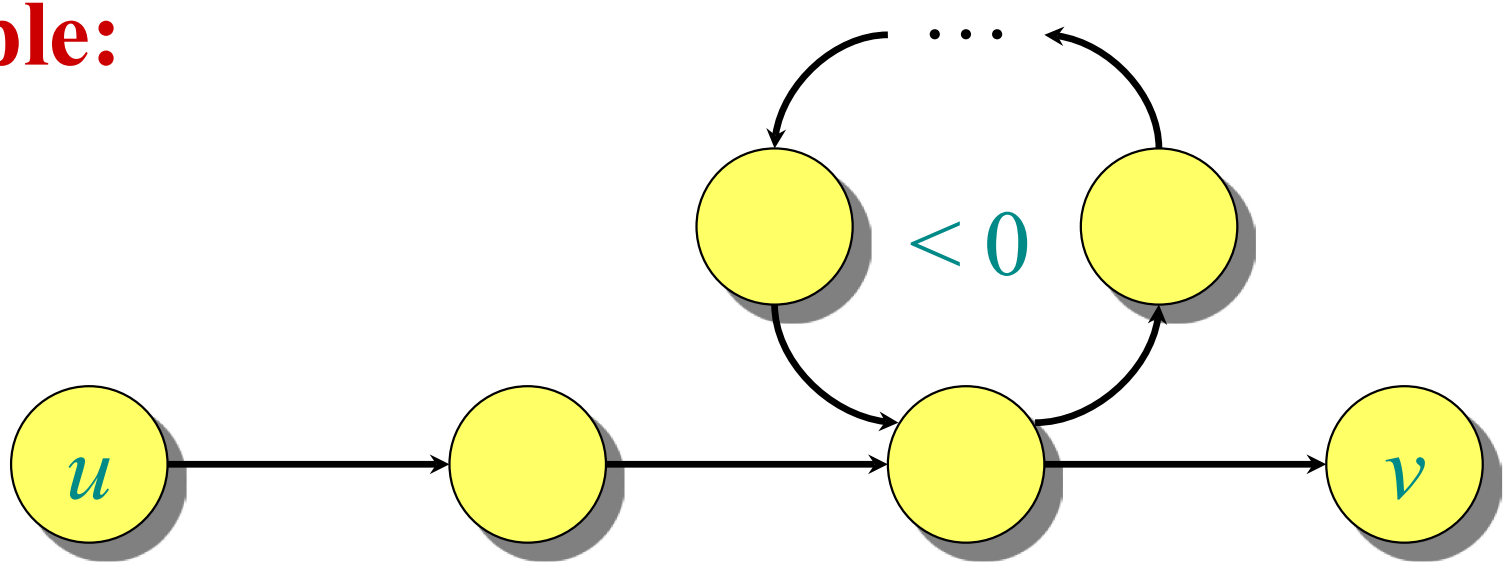




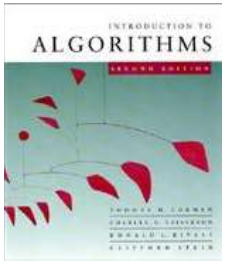
Negative-weight cycles

Recall: If a graph $G = (V, E)$ contains a negative-weight cycle, then some shortest paths may not exist.

Example:



Bellman-Ford algorithm: Finds all shortest-path lengths from a ***source*** $s \in V$ to all $v \in V$ or determines that a negative-weight cycle exists.



Bellman-Ford algorithm

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
  do $d[v] \leftarrow \infty$

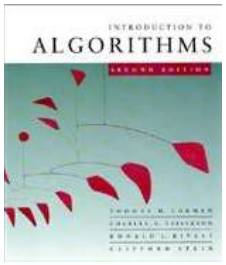
} initialization

for $i \leftarrow 1$ **to** $|V| - 1$
  do for each edge $(u, v) \in E$
    do if $d[v] > d[u] + w(u, v)$
      then $d[v] \leftarrow d[u] + w(u, v)$

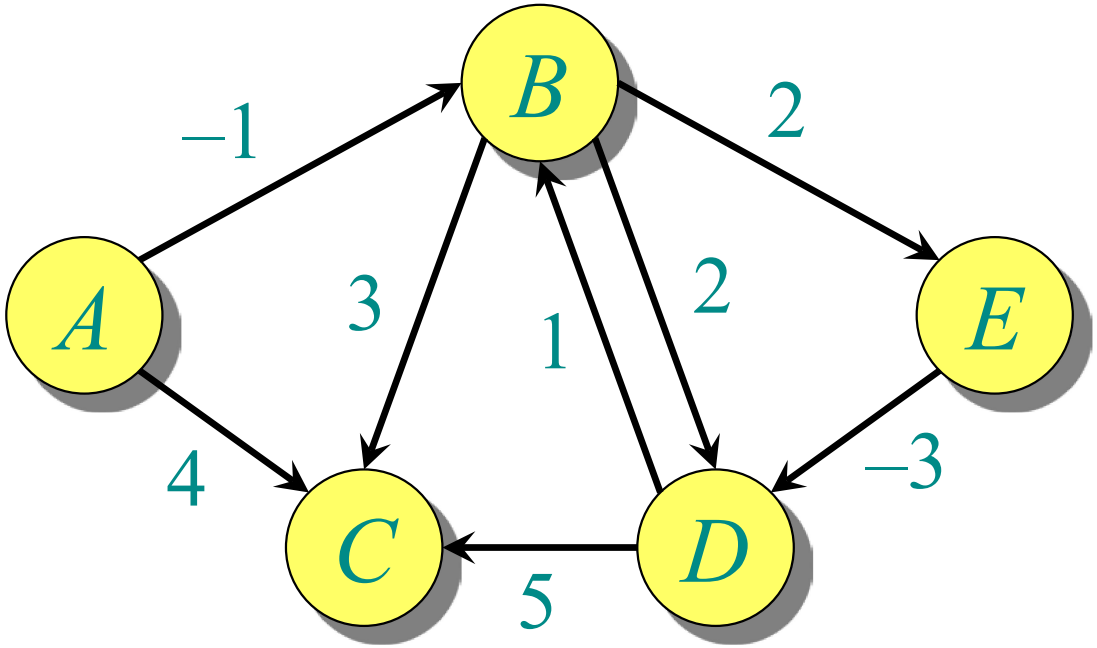
} ***relaxation step***

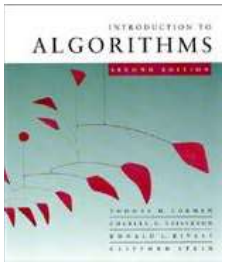
for each edge $(u, v) \in E$
  do if $d[v] > d[u] + w(u, v)$
    then report that a negative-weight cycle exists

At the end, $d[v] = \delta(s, v)$, if no negative-weight cycles.
Time = $O(VE)$.

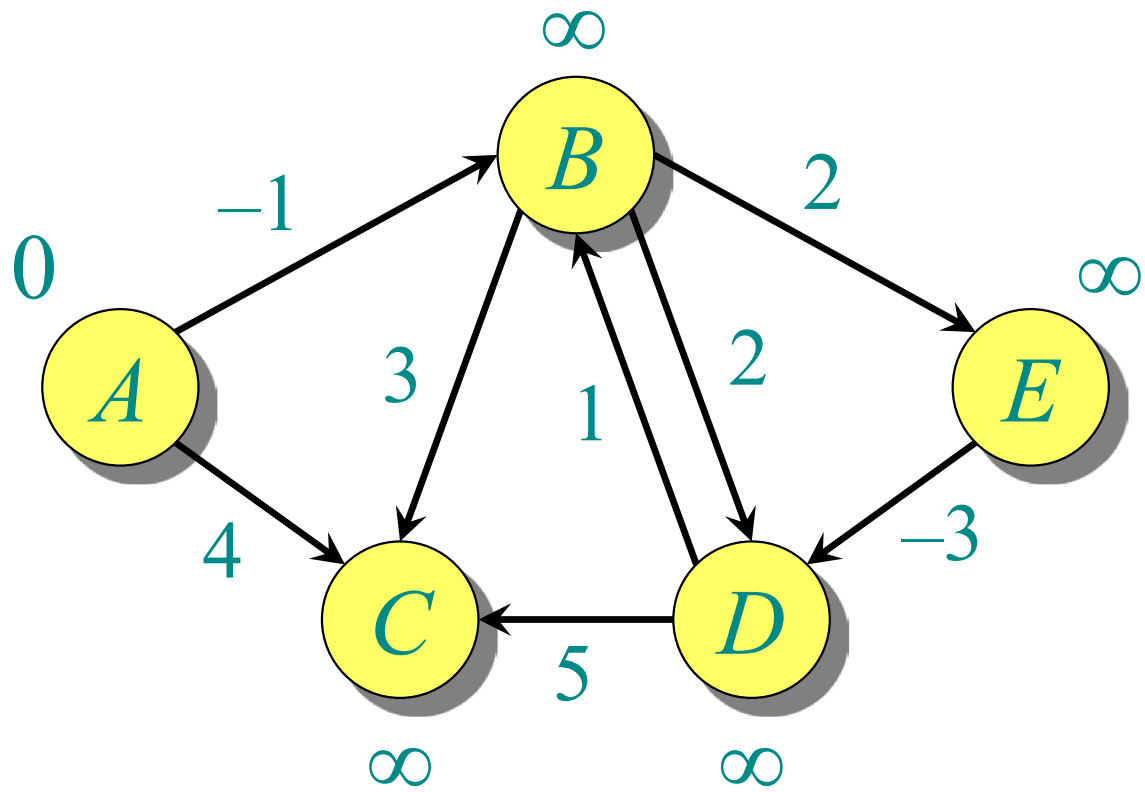


Example of Bellman-Ford

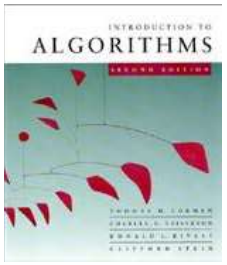




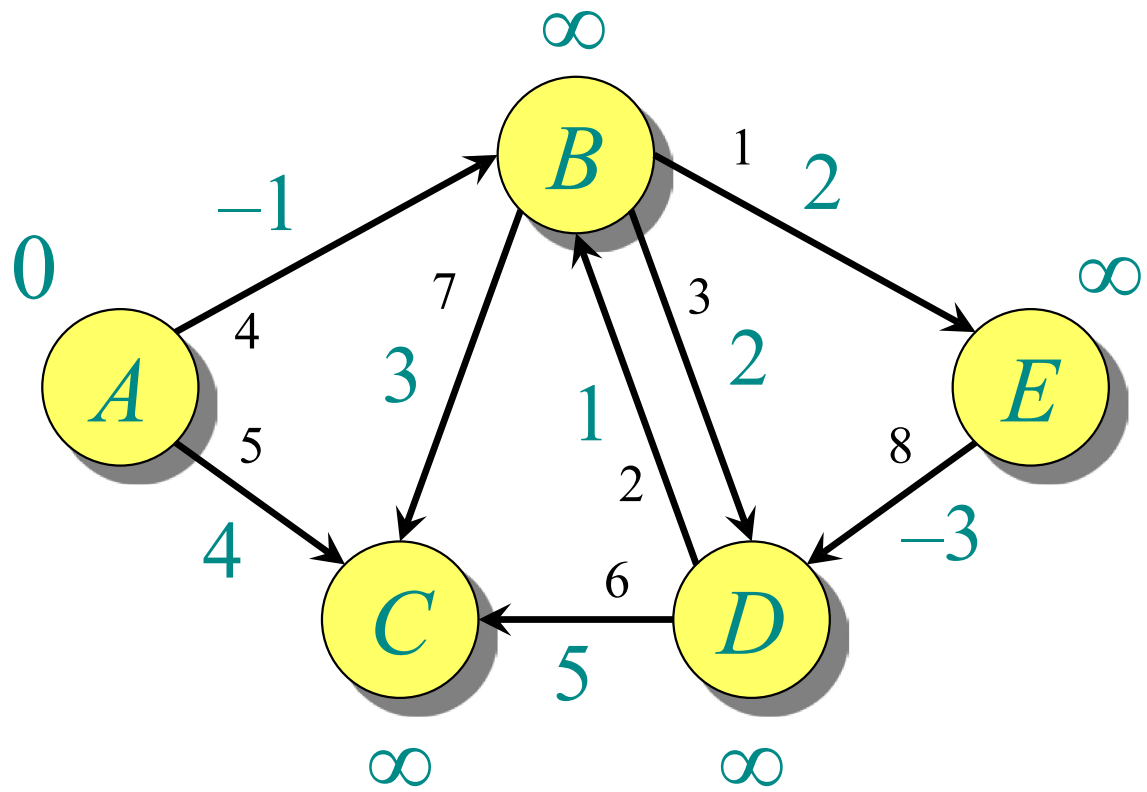
Example of Bellman-Ford



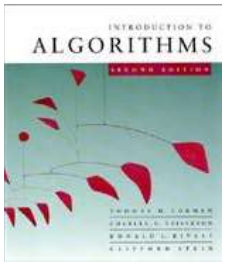
Initialization.



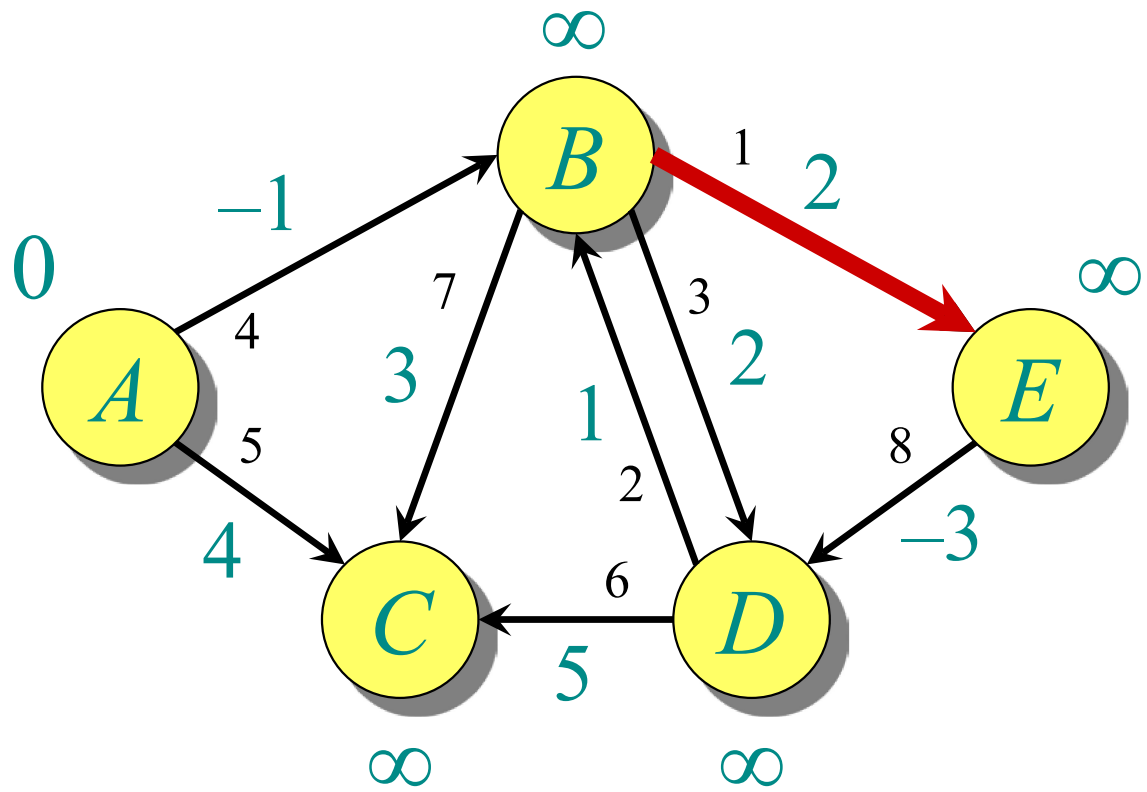
Example of Bellman-Ford

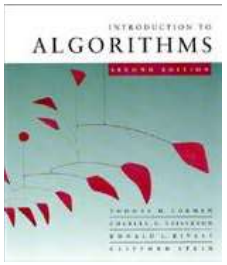


Order of edge relaxation.

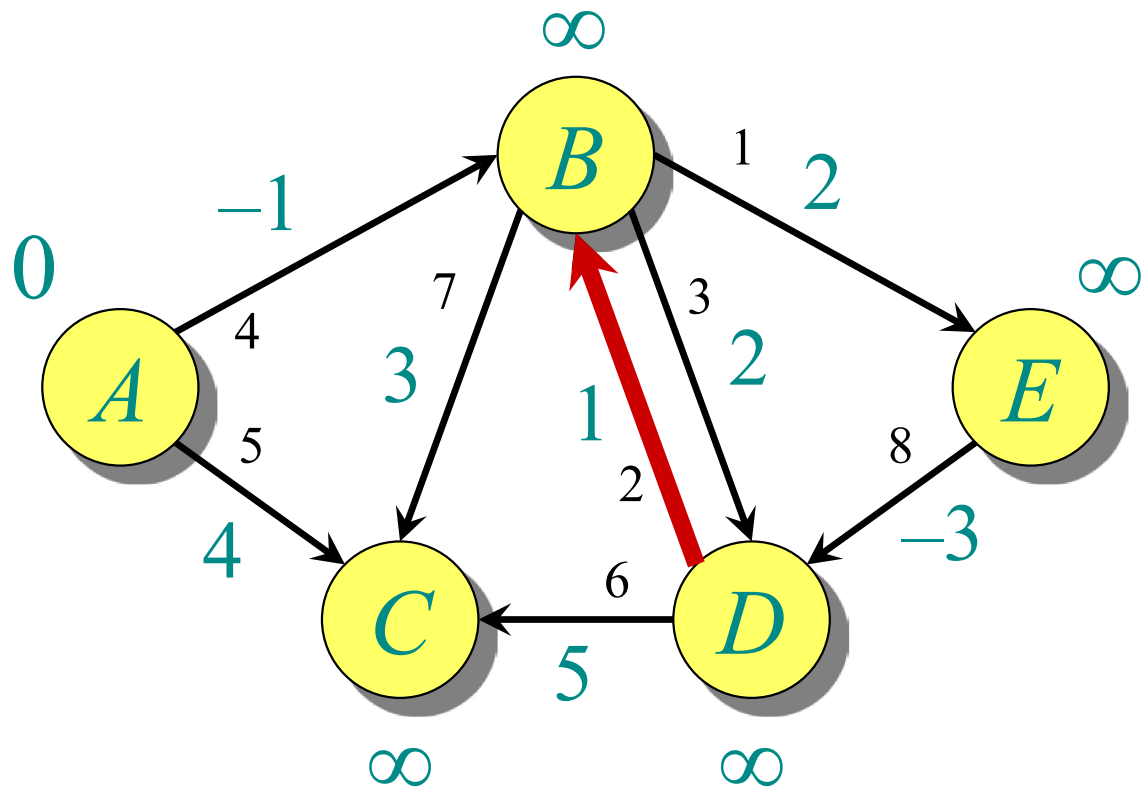


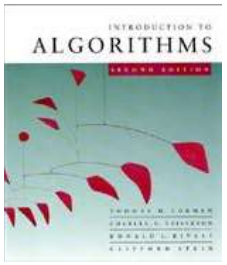
Example of Bellman-Ford



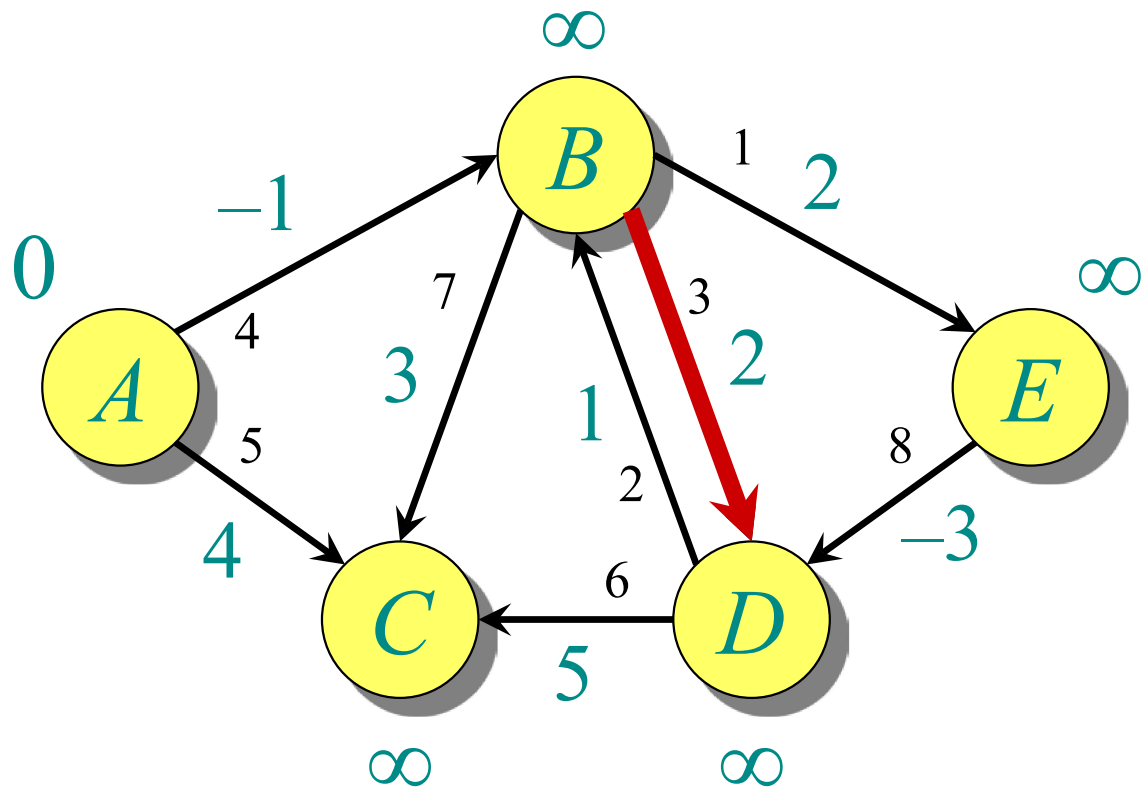


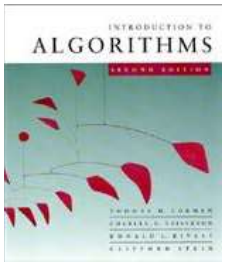
Example of Bellman-Ford



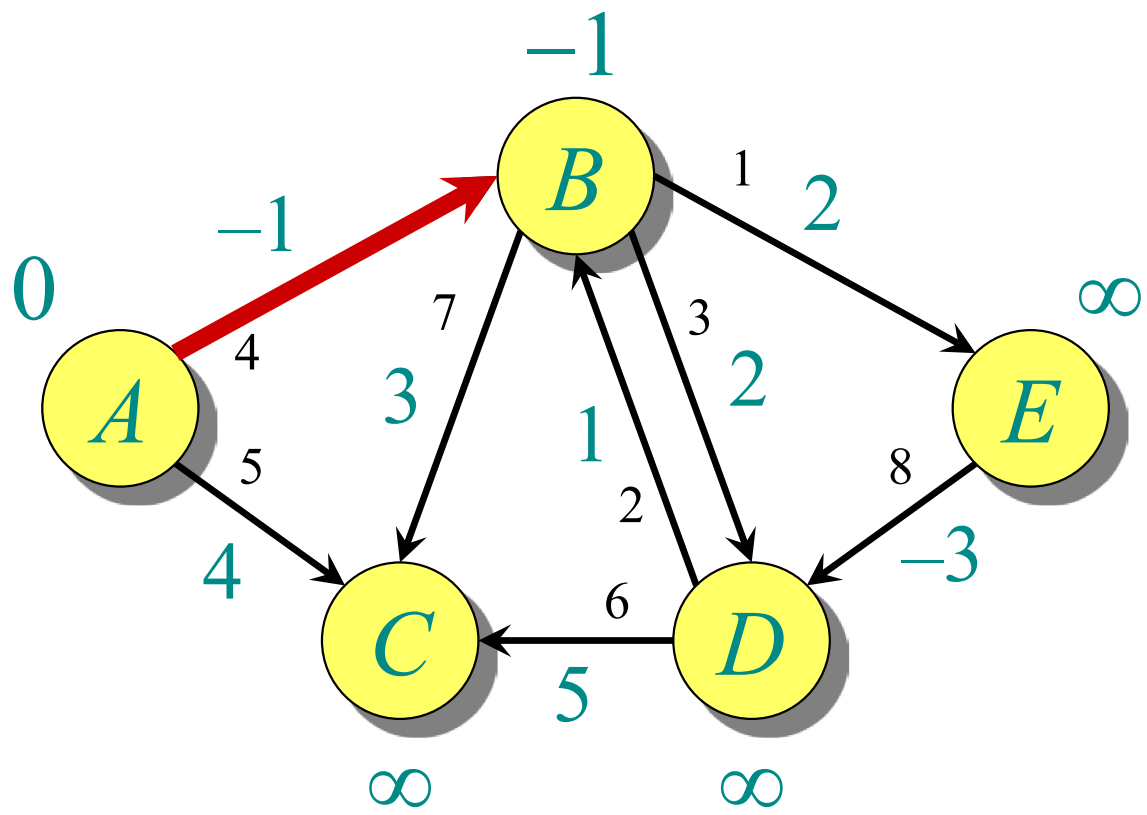


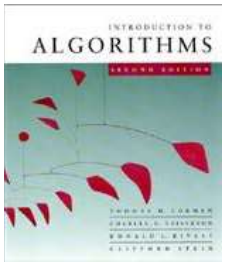
Example of Bellman-Ford



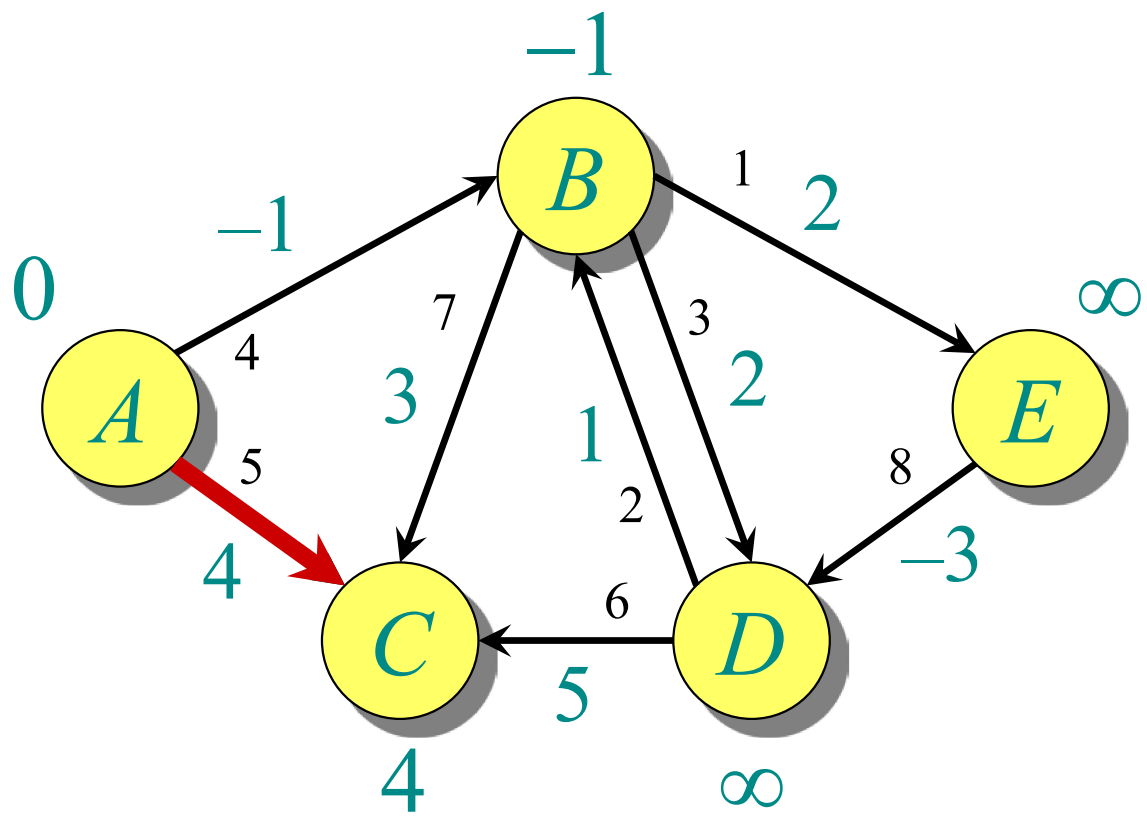


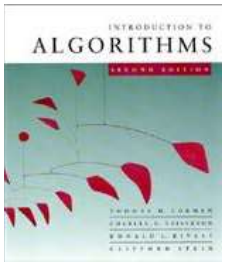
Example of Bellman-Ford



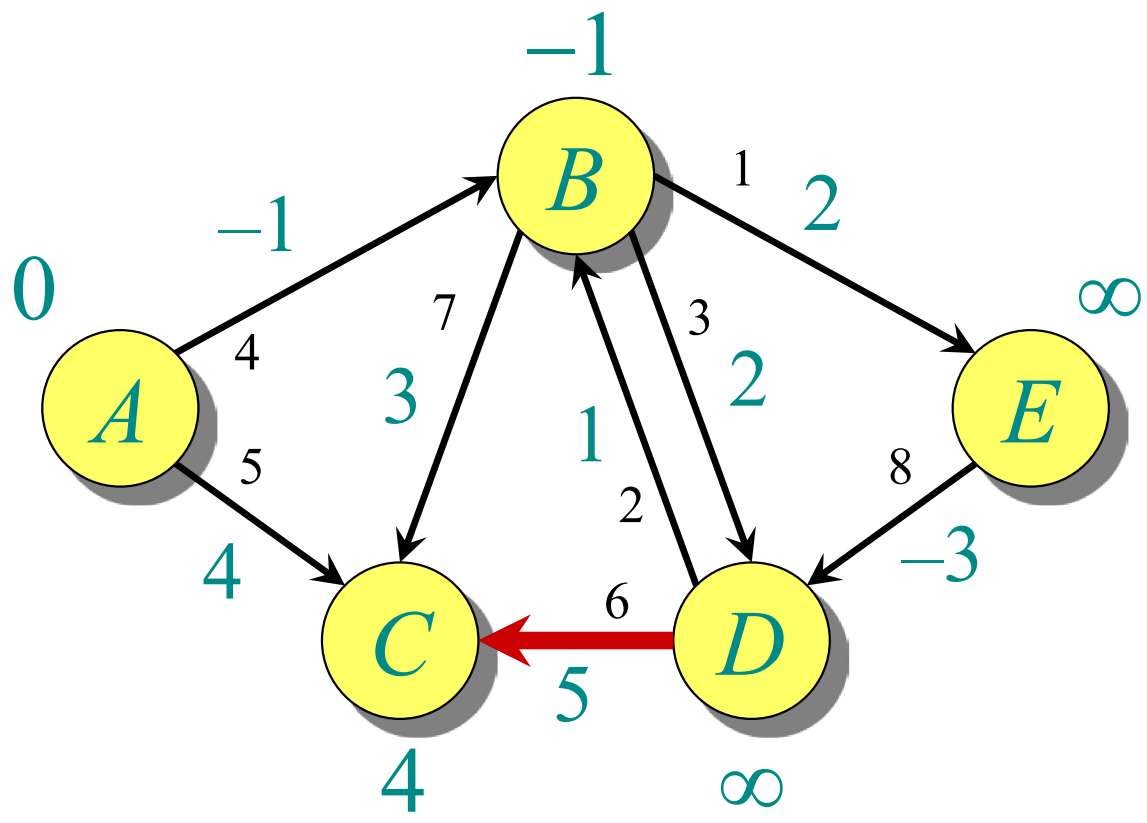


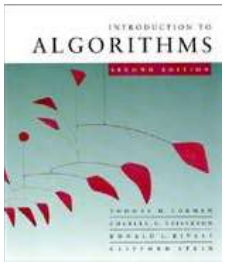
Example of Bellman-Ford



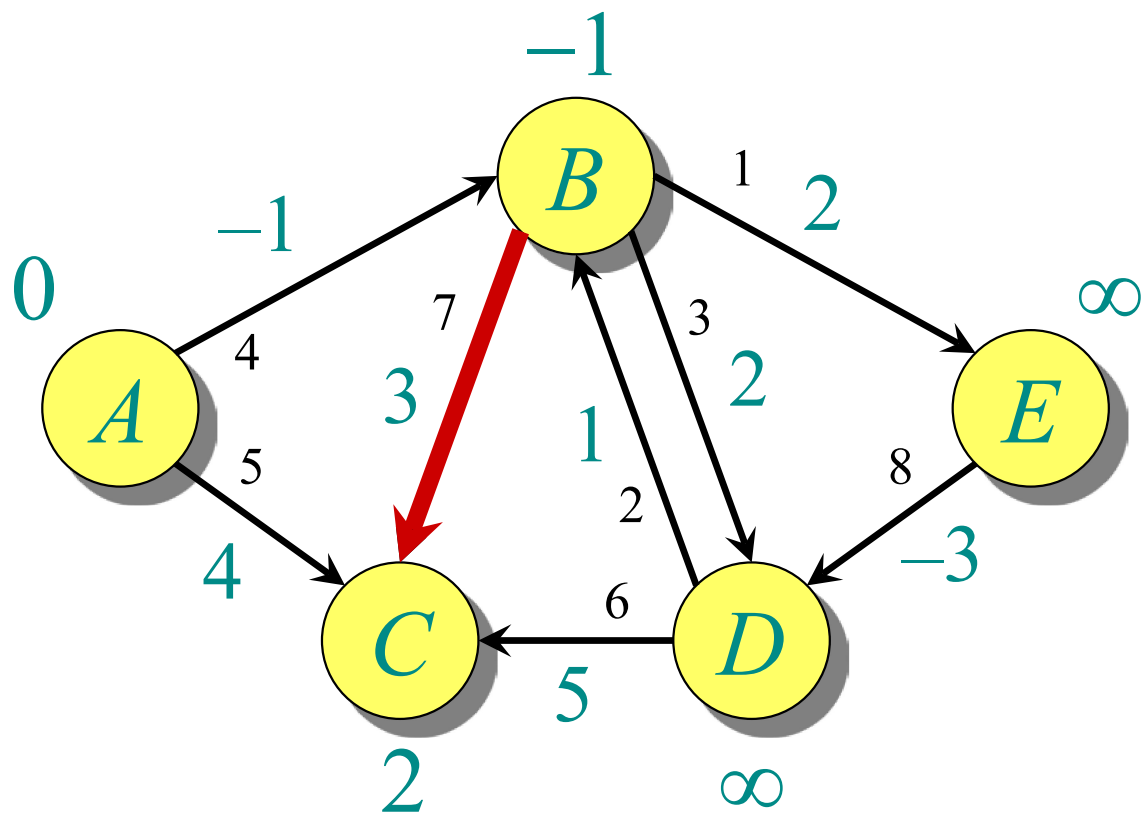


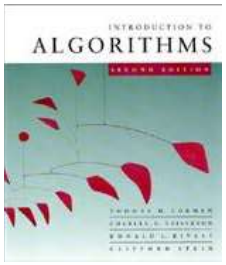
Example of Bellman-Ford



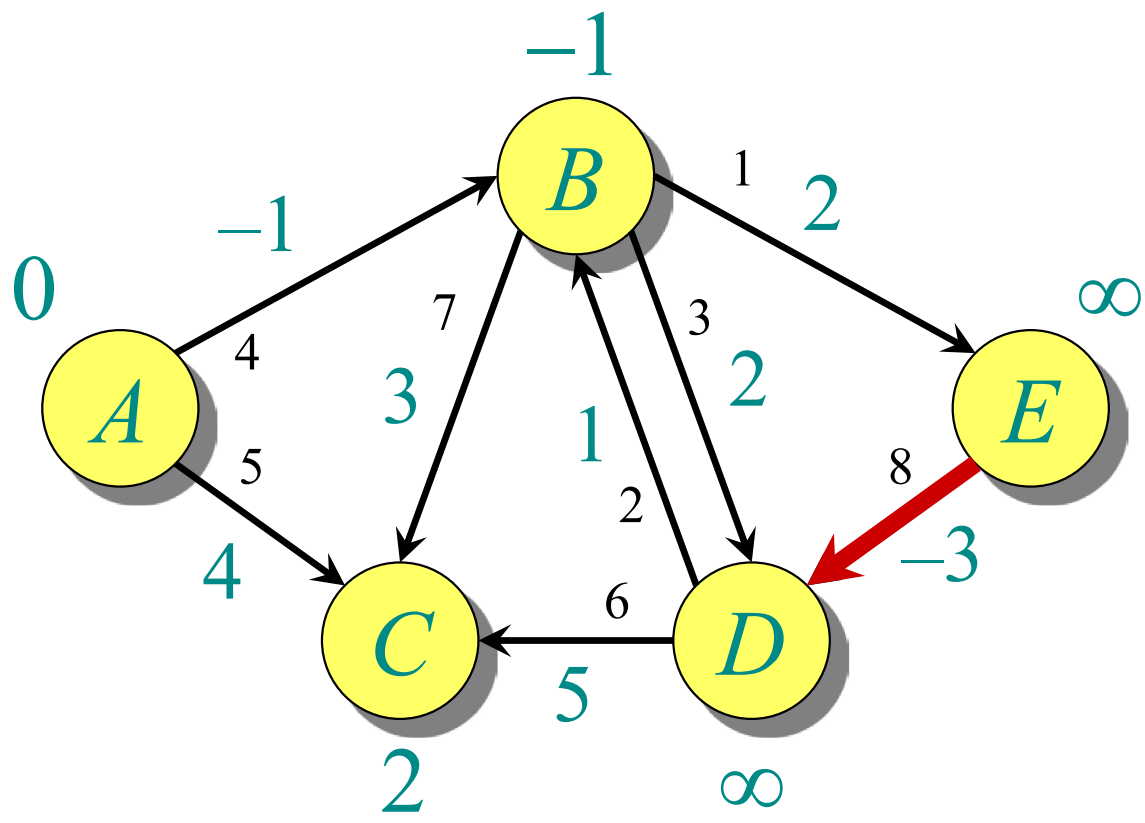


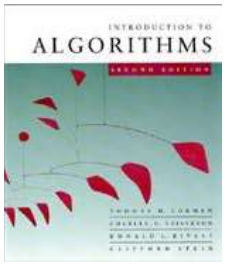
Example of Bellman-Ford



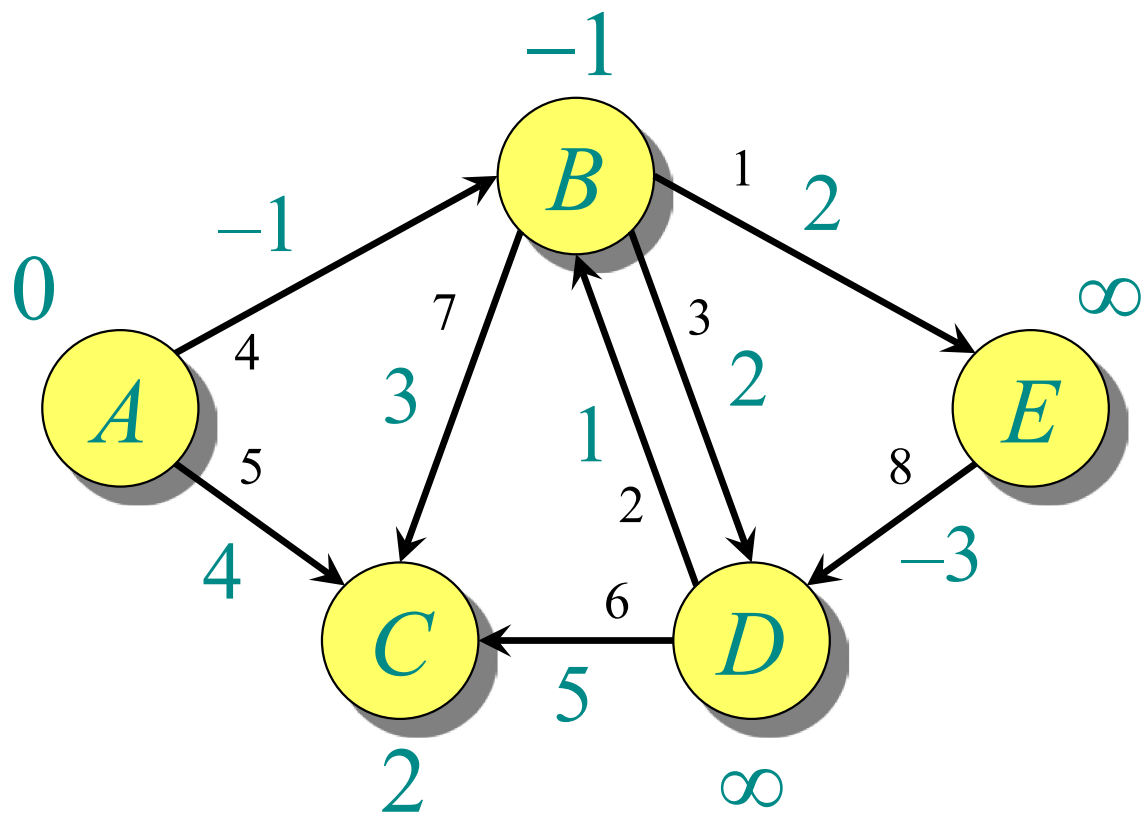


Example of Bellman-Ford

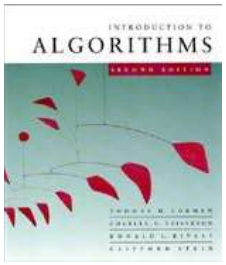




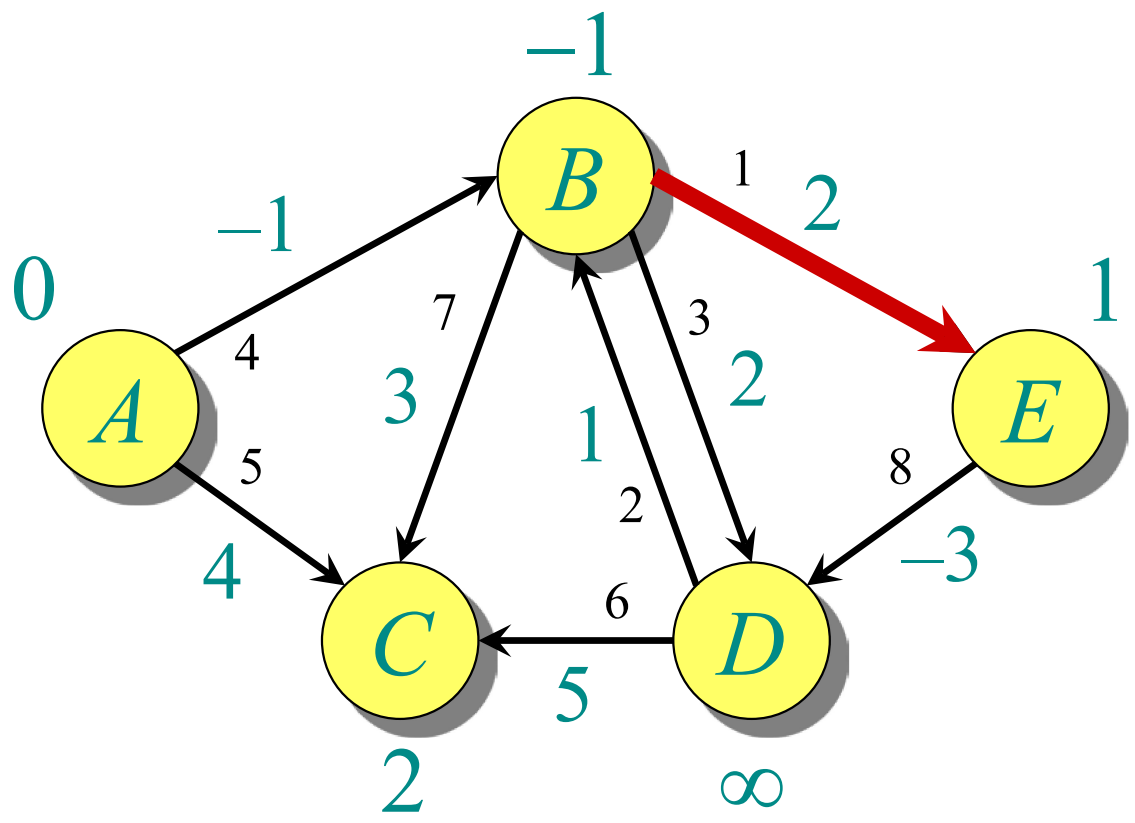
Example of Bellman-Ford

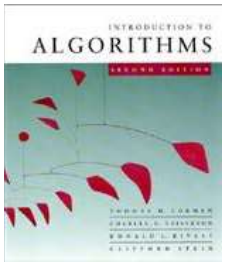


End of pass 1.

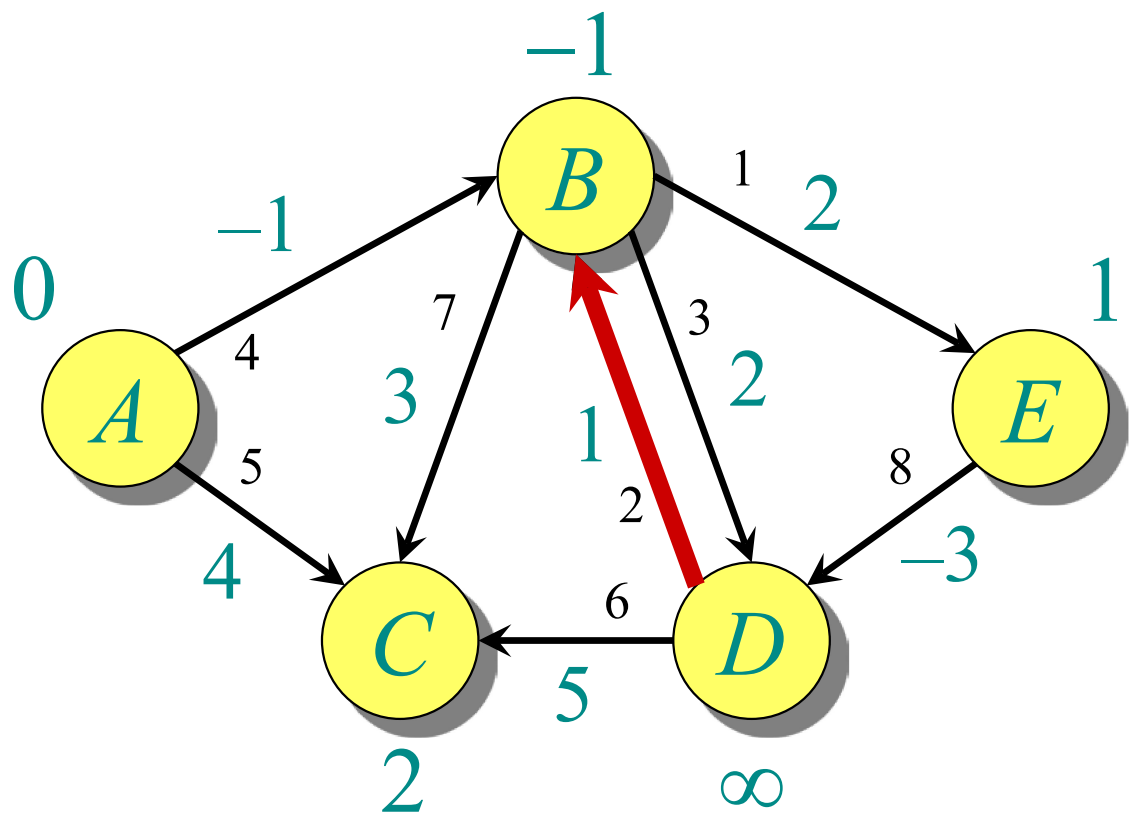


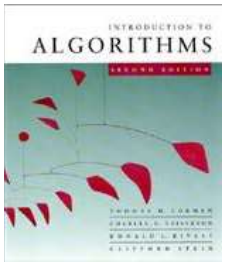
Example of Bellman-Ford



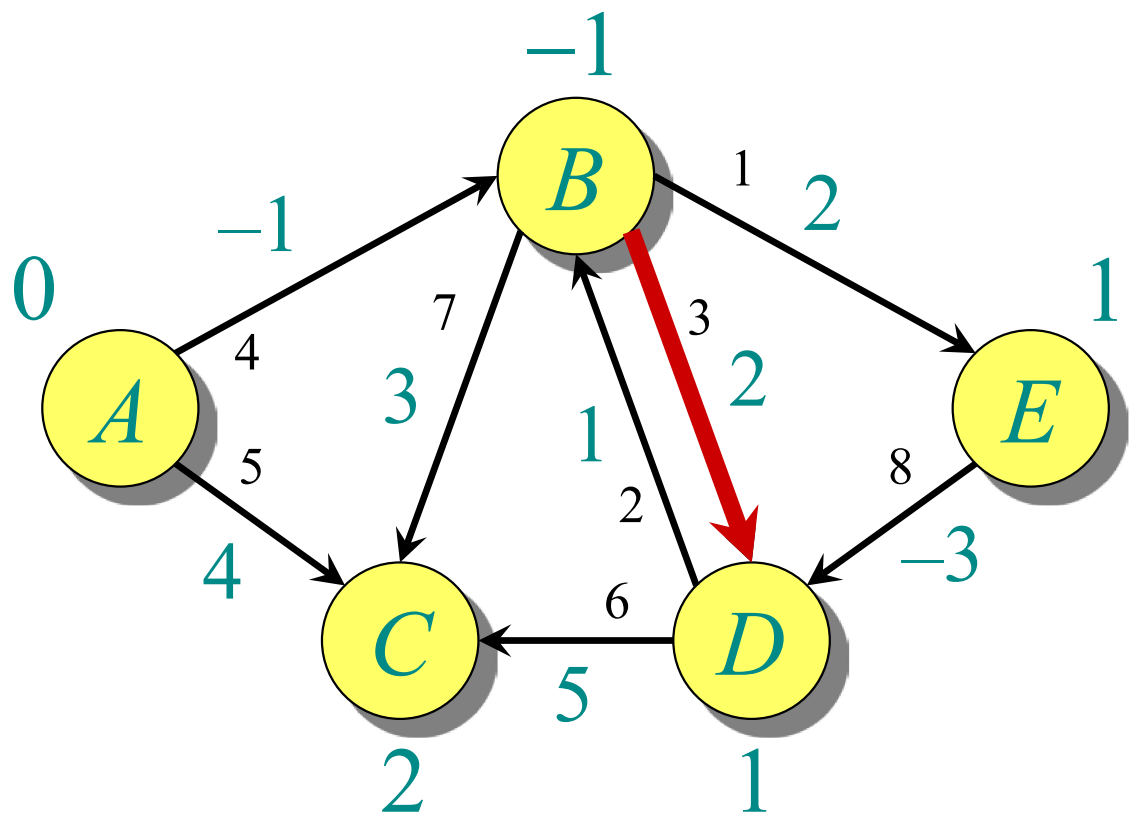


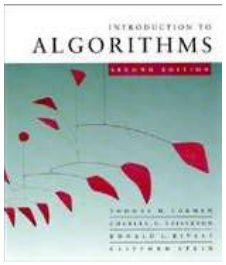
Example of Bellman-Ford



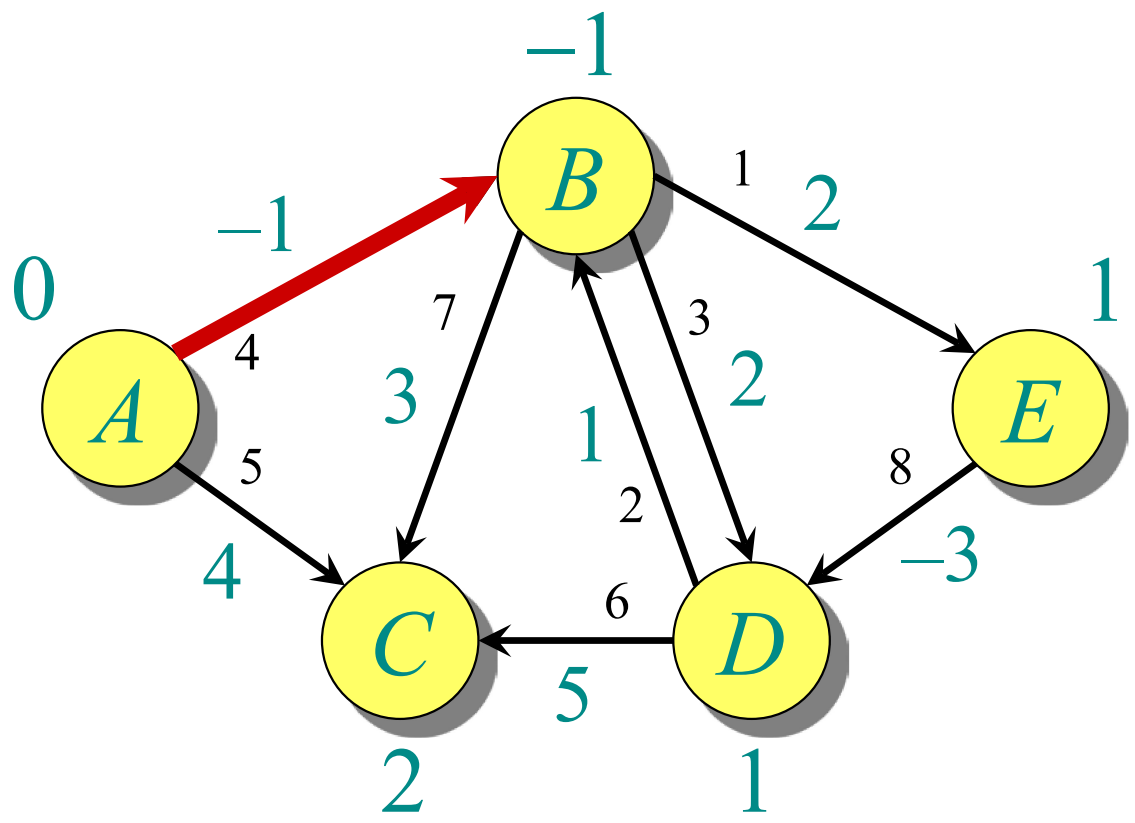


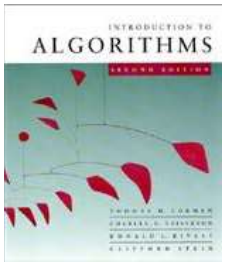
Example of Bellman-Ford



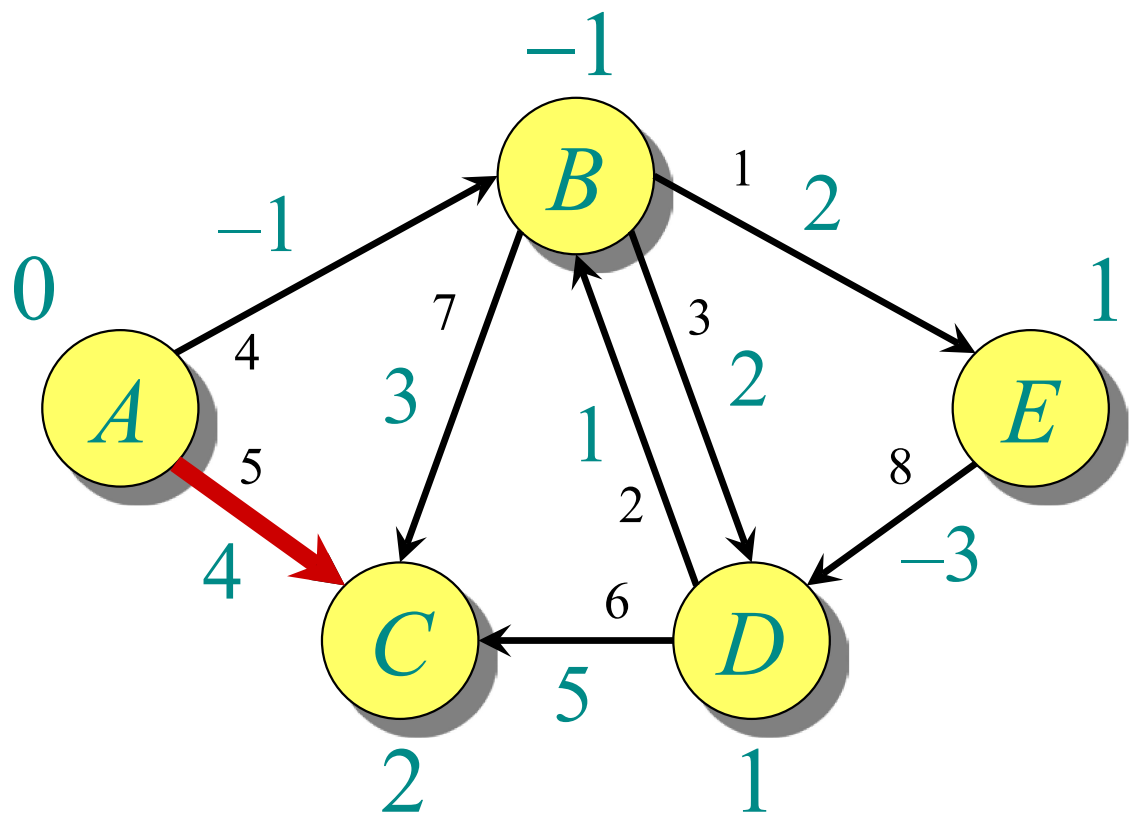


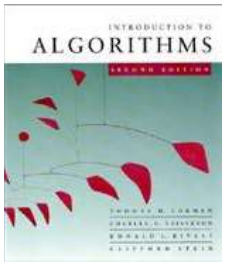
Example of Bellman-Ford



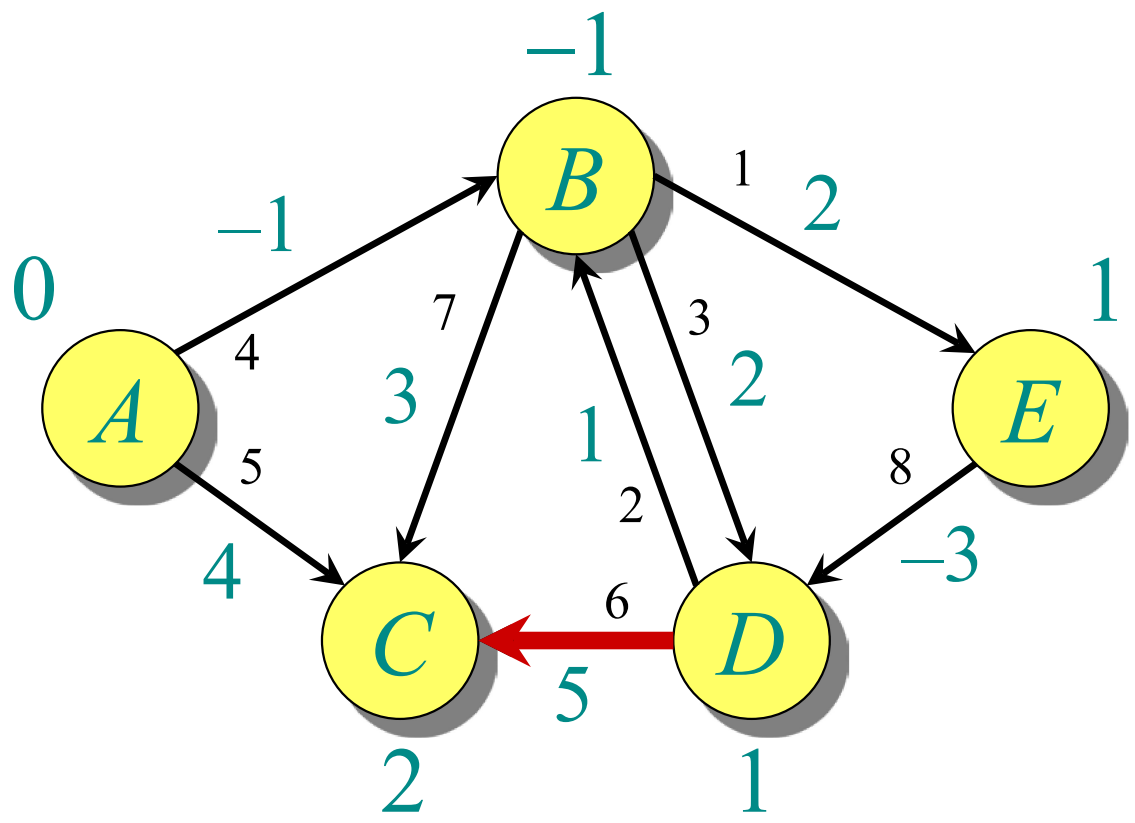


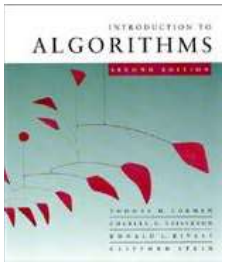
Example of Bellman-Ford



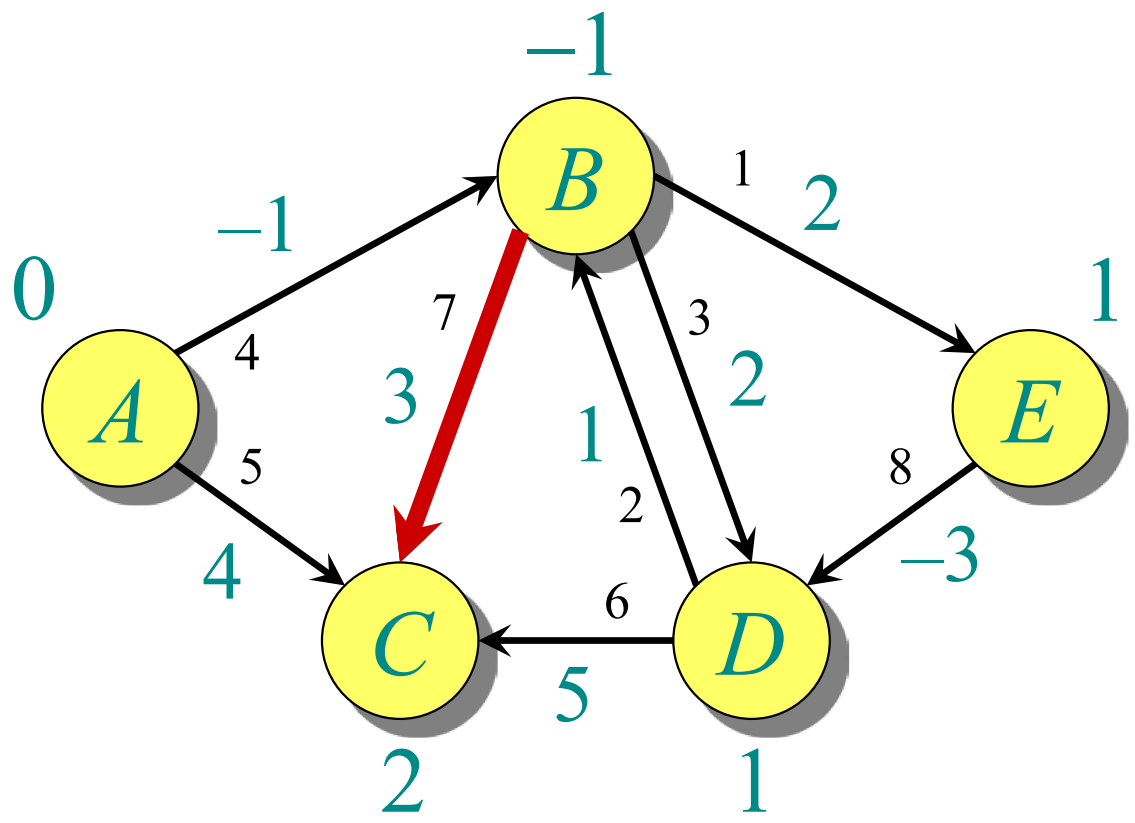


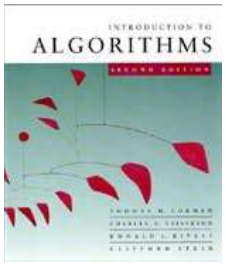
Example of Bellman-Ford



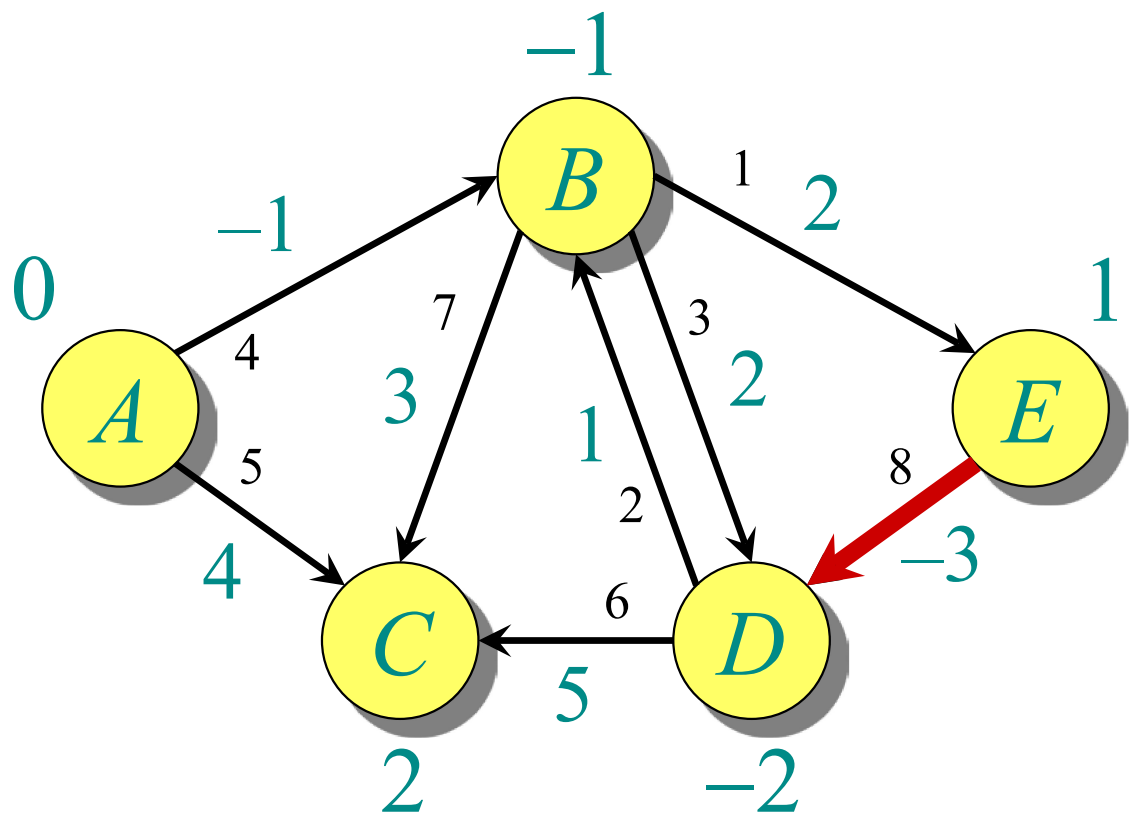


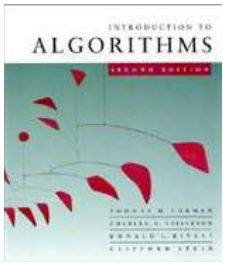
Example of Bellman-Ford



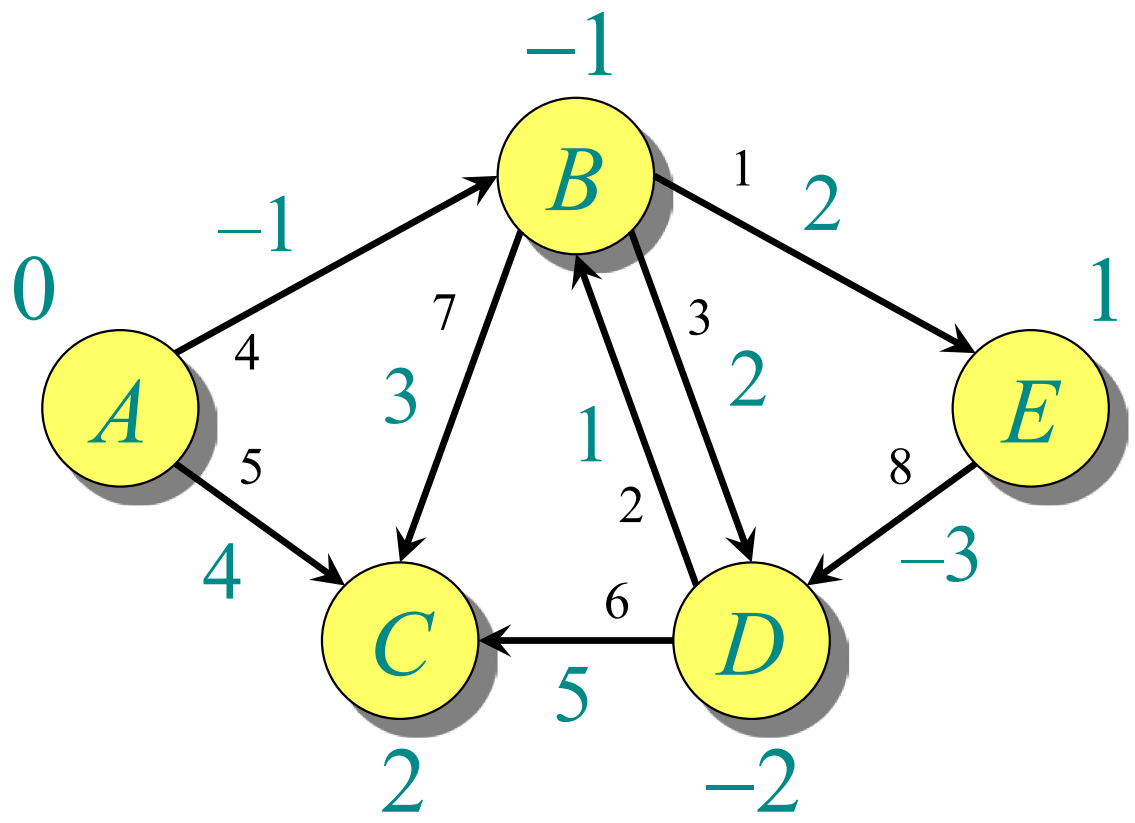


Example of Bellman-Ford

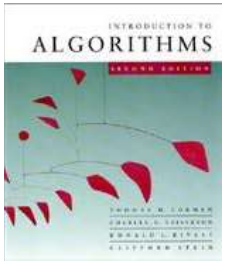




Example of Bellman-Ford

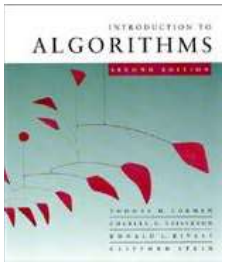


End of pass 2 (and 3 and 4).



Correctness

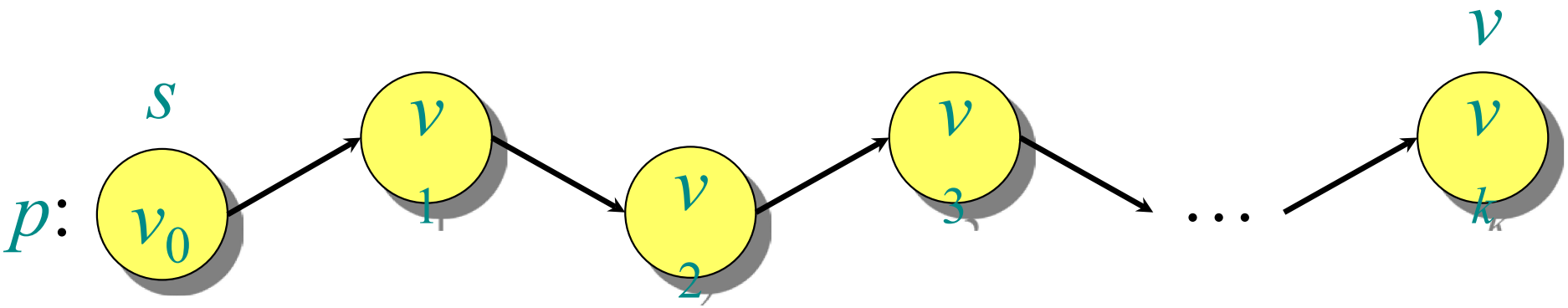
Theorem. If $G = (V, E)$ contains no negative-weight cycles, then after the Bellman-Ford algorithm executes, $d[v] = \delta(s, v)$ for all $v \in V$.



Correctness

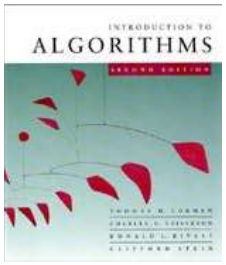
Theorem. If $G = (V, E)$ contains no negative-weight cycles, then after the Bellman-Ford algorithm executes, $d[v] = \delta(s, v)$ for all $v \in V$.

Proof. Let $v \in V$ be any vertex, and consider a shortest path p from s to v with the minimum number of edges.

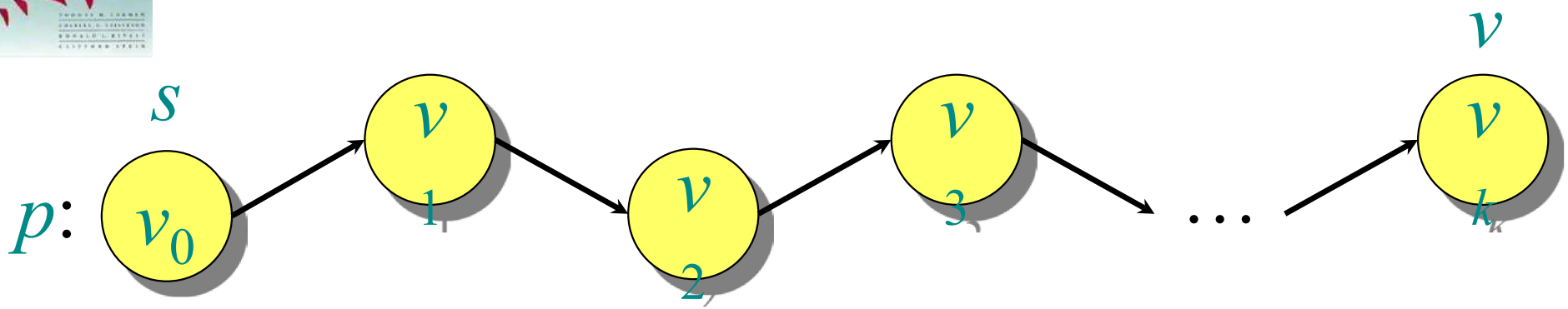


Since p is a shortest path, we have

$$\delta(s, v_i) = \delta(s, v_{i-1}) + w(v_{i-1}, v_i) \ .$$



Correctness (continued)

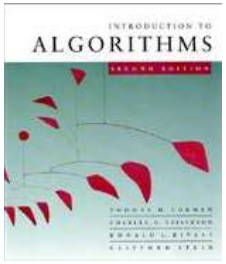


Initially, $d[v_0] = 0 = \delta(s, v_0)$, and $d[v_0]$ is unchanged by subsequent relaxations (because of the lemma from *Shortest Paths I* that $d[v] \geq \delta(s, v)$).

- After 1 pass through E, we have $d[v_1] = \delta(s, v_1)$.
- After 2 passes through E, we have $d[v_2] = \delta(s, v_2)$.

  ⋮
- After k passes through E, we have $d[v_k] = \delta(s, v_k)$.

Since G contains no negative-weight cycles, p is simple. Longest simple path has $\leq |V| - 1$ edges. □



Detection of negative-weight cycles

Corollary. If a value $d[v]$ fails to converge after $|V| - 1$ passes, there exists a negative-weight cycle in G reachable from s. □

