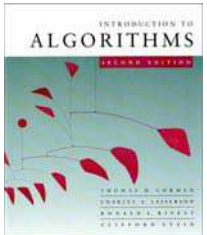
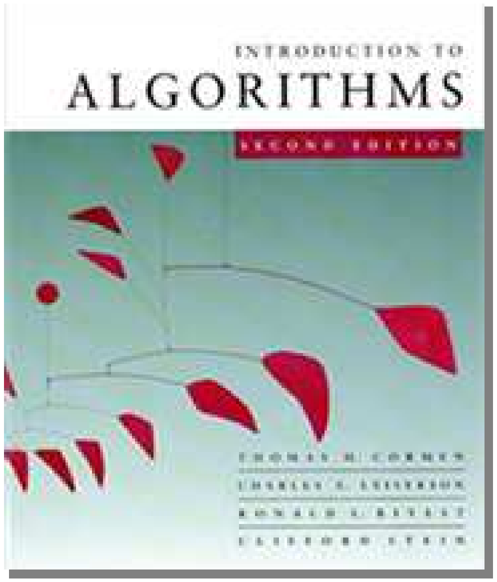


Introduction to Algorithms

6.046J/18.401J

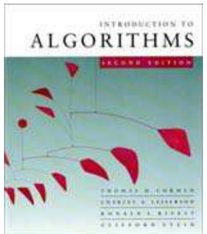


LECTURE 17

Shortest Paths I

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

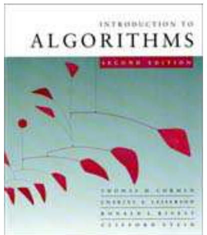
Prof. Erik Demaine



Paths in graphs

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

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

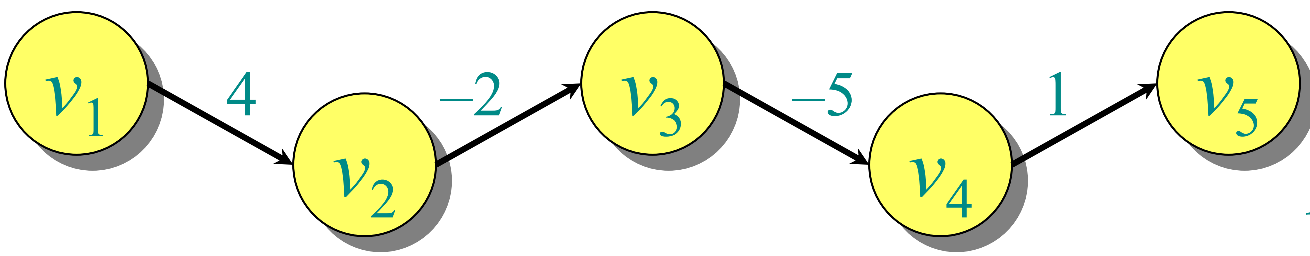


Paths in graphs

Consider a digraph $G = (V, E)$ with edge-weight function $w : E \to \mathbb{R}$. 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

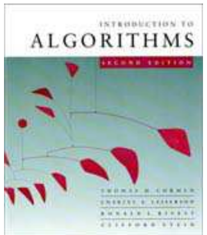
A ***shortest path*** from u to v is a path of minimum weight from u to v. The ***shortest-path weight*** from u to v is defined as

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

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

Optimal substructure

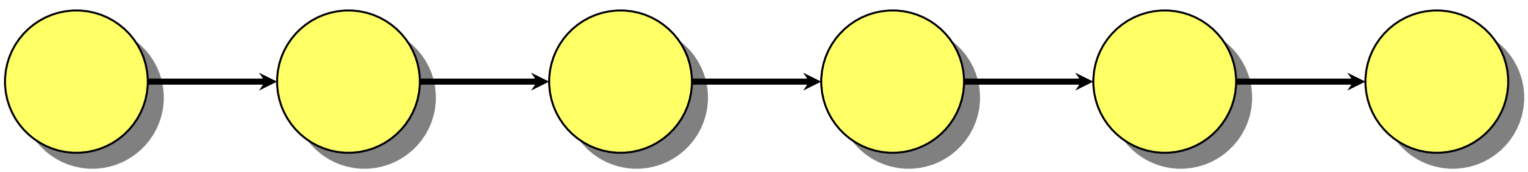
Theorem. A subpath of a shortest path is a shortest path.



Optimal substructure

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

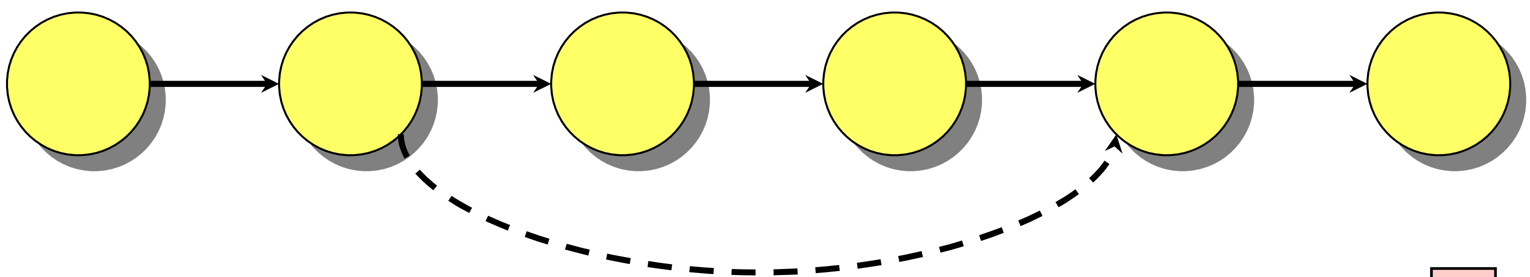
Proof. Cut and paste:



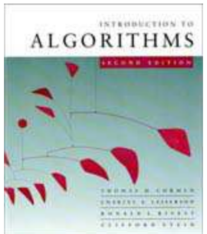
Optimal substructure

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

Proof. Cut and paste:



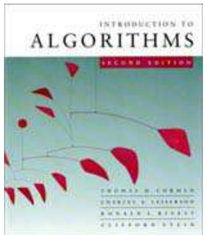
∎



Triangle inequality

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

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

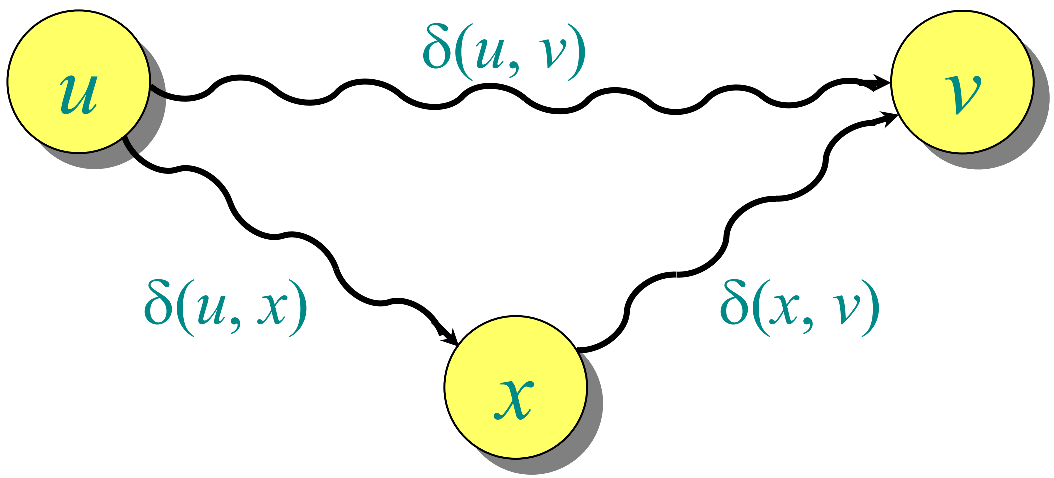


Triangle inequality

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

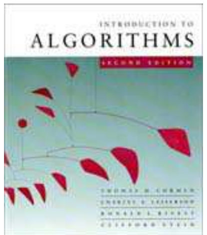
$$\delta(u, v) \leq \delta(u, x) + \delta(x, v).$$

Proof.



Well-definedness of shortest paths

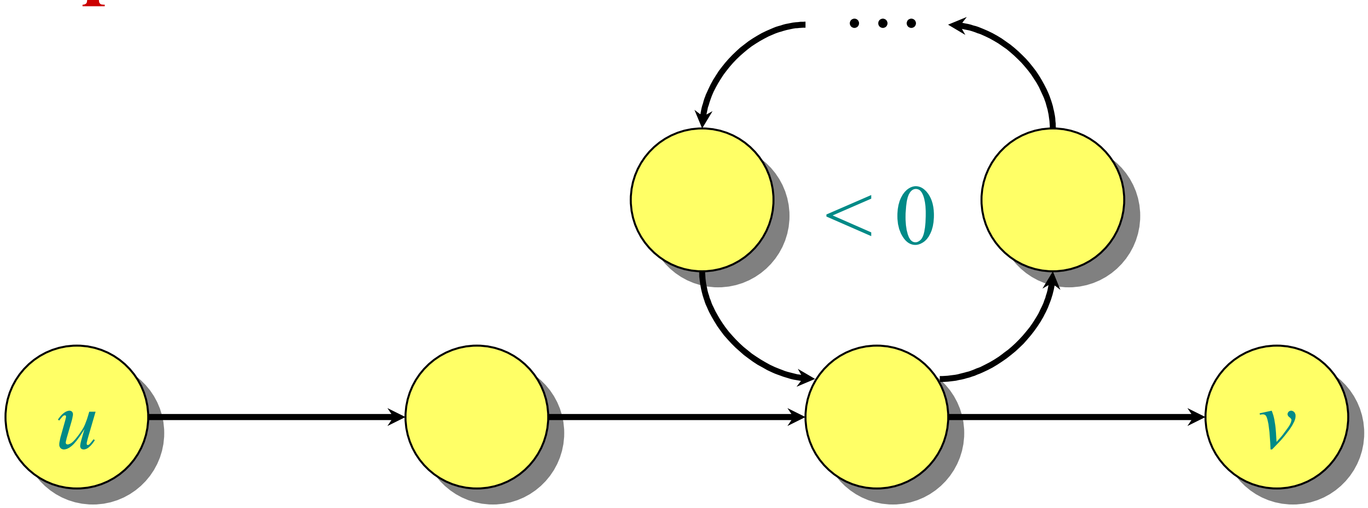
If a graph *G* contains a negative-weight cycle, then some shortest paths may not exist.

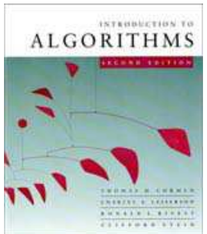


Well-definedness of shortest paths

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

Example:





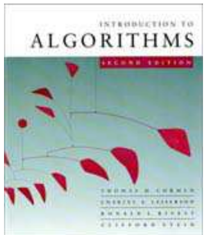
Single-source shortest paths

Problem. From a given source vertex $s \in V$, find the shortest-path weights $\delta(s, v)$ for all $v \in V$.

If all edge weights $w(u, v)$ are *nonnegative*, all shortest-path weights must exist.

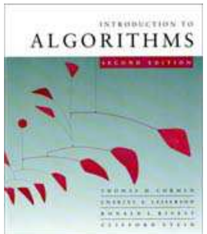
IDEA: Greedy.

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 minimal.
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 $V - S$



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 $V - S$

while $Q \neq \varnothing$

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

    if d[u] = infinity: break;

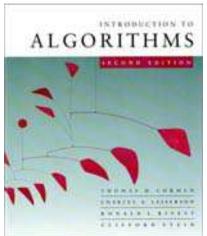
    $S \leftarrow S \cup \{u\}$

    for each $v \in Adj[u]$ (with v from Q)

      do if $d[v] > d[u] + w(u, v)$

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

NOTE: all remaining vertices are not accessible from source



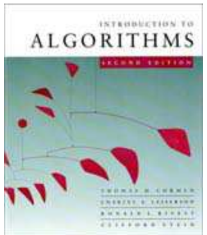
Dijkstra's algorithm

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

NOTE: all remaining vertices are not accessible from source

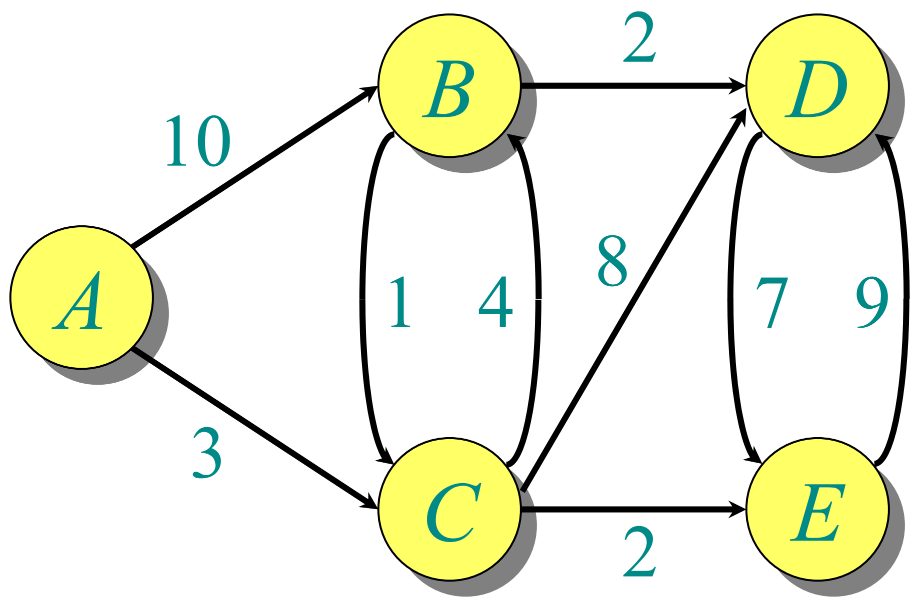
relaxation step

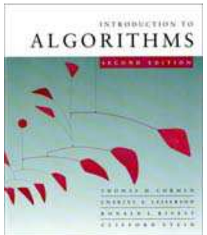
Implicit DECREASE-KEY



Example of Dijkstra's algorithm

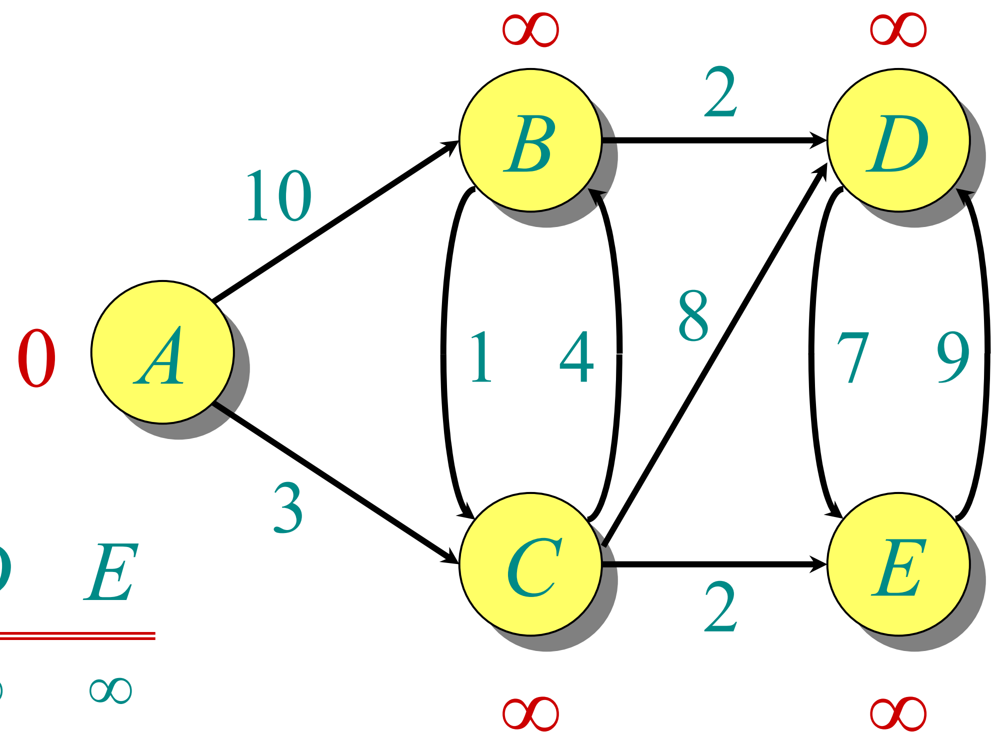
Graph with nonnegative edge weights:





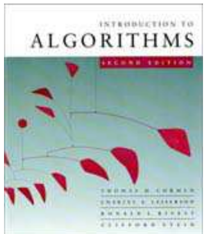
Example of Dijkstra's algorithm

Initialize:



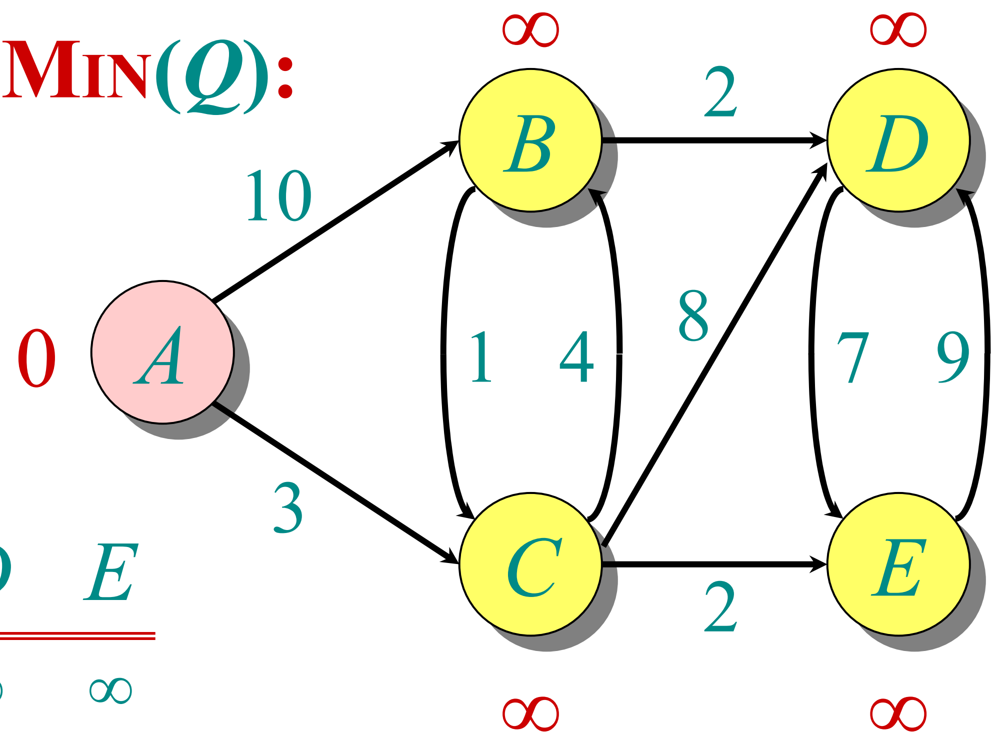
Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: {}



Example of Dijkstra's algorithm

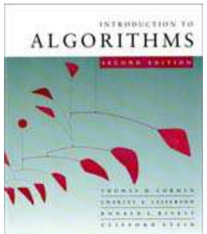
"A" ← EXTRACT-MIN(Q):



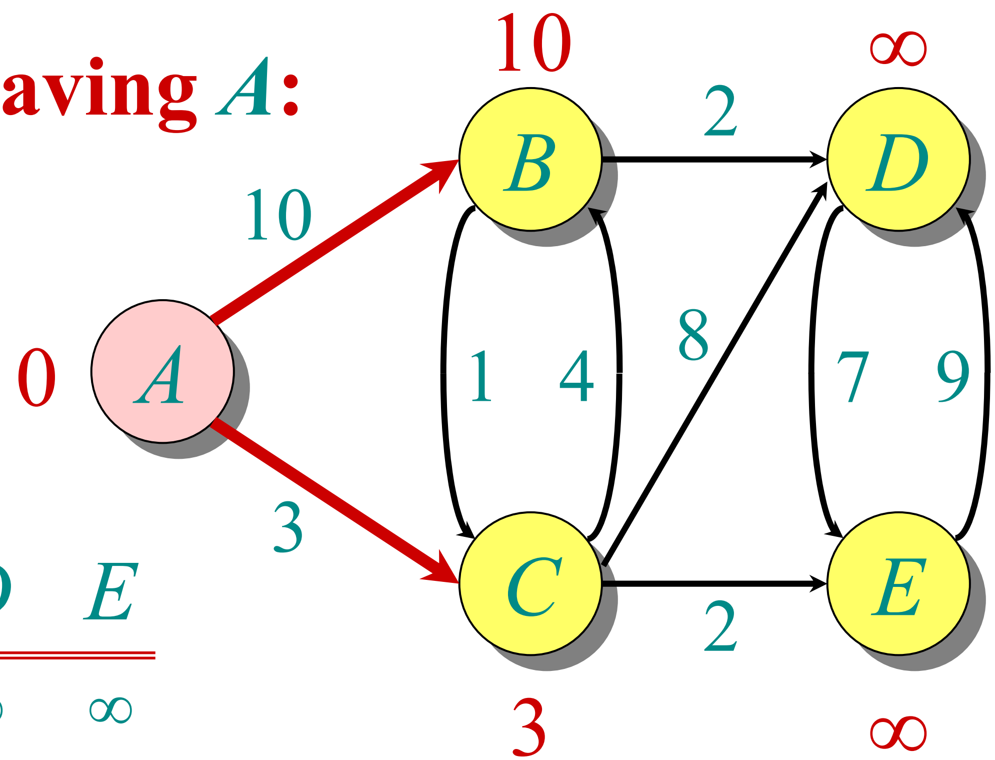
Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: { A }

Example of Dijkstra's algorithm

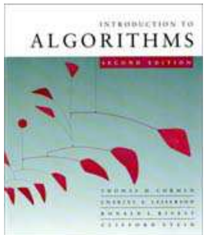


Relax all edges leaving *A*:



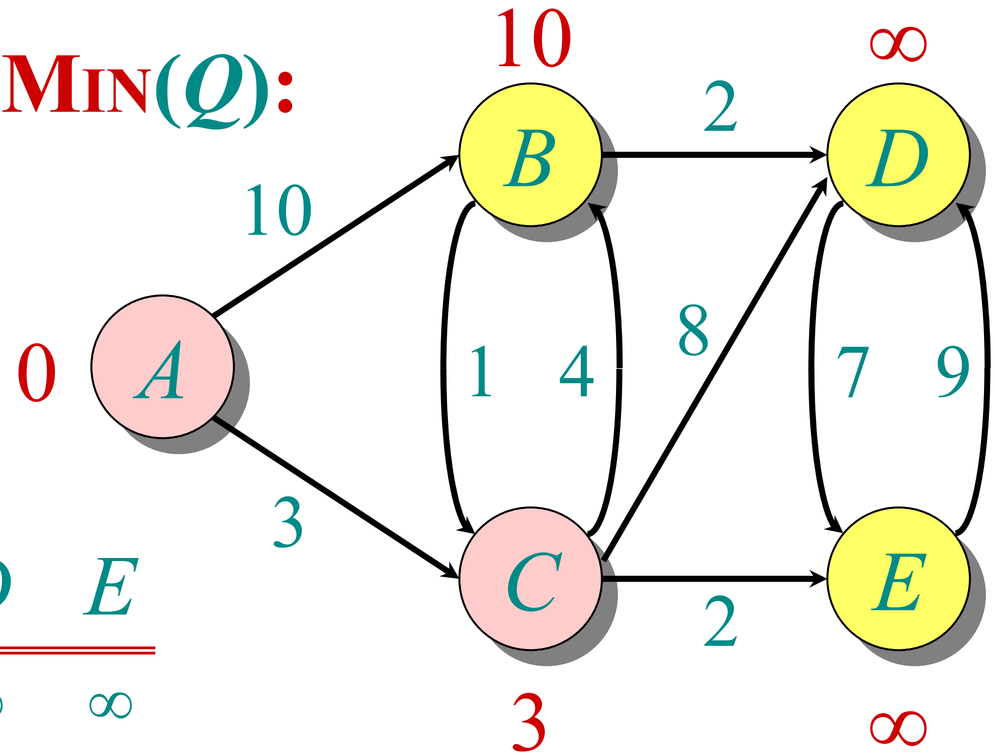
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { *A* }



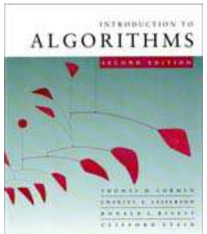
Example of Dijkstra's algorithm

"C" ← **EXTRACT-MIN(*Q*):**



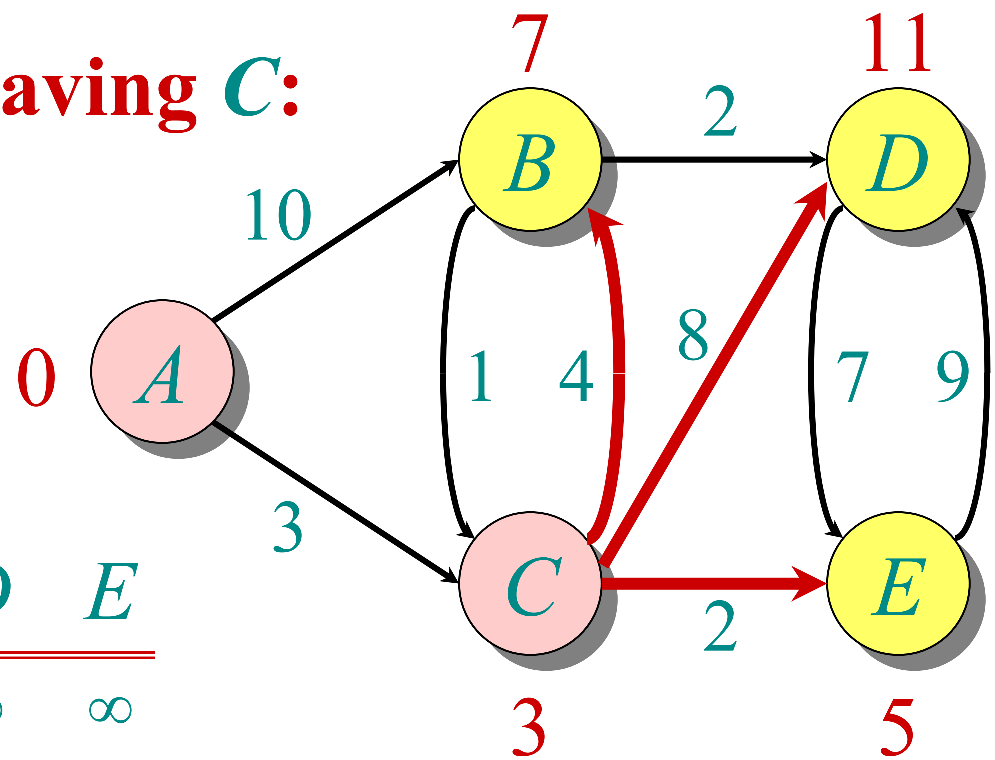
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { *A*, *C* }



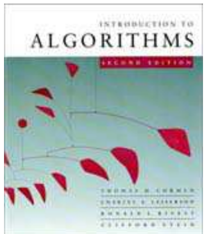
Example of Dijkstra's algorithm

Relax all edges leaving *C*:



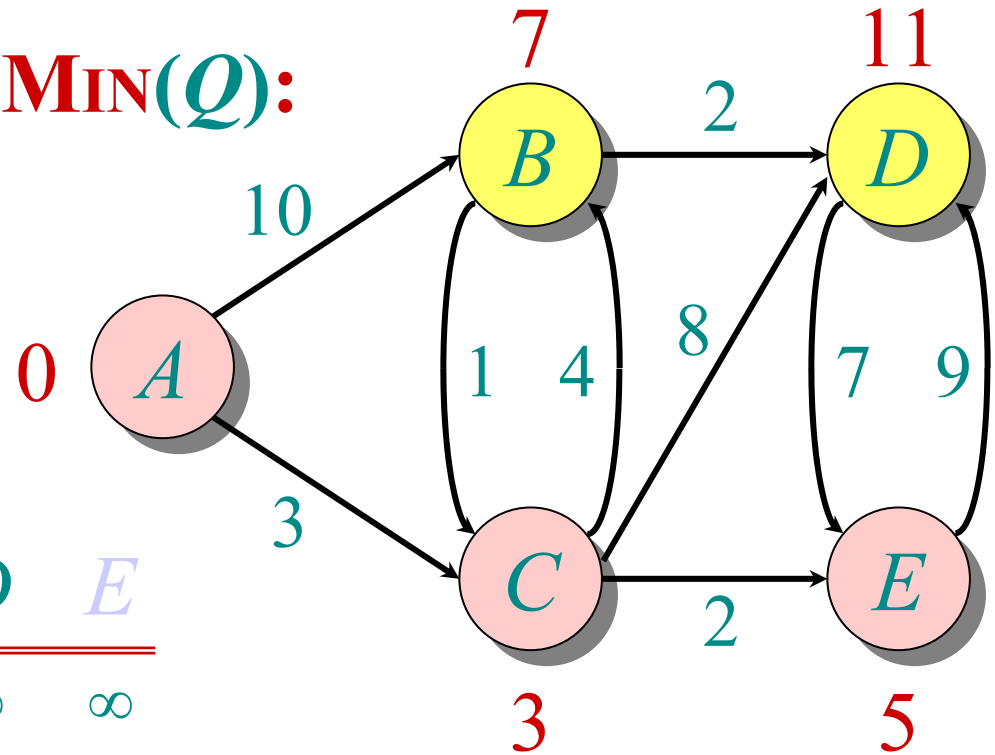
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { *A*, *C* }



Example of Dijkstra's algorithm

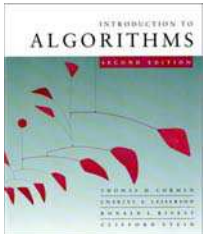
"*E*" ← EXTRACT-MIN(*Q*):



Q:

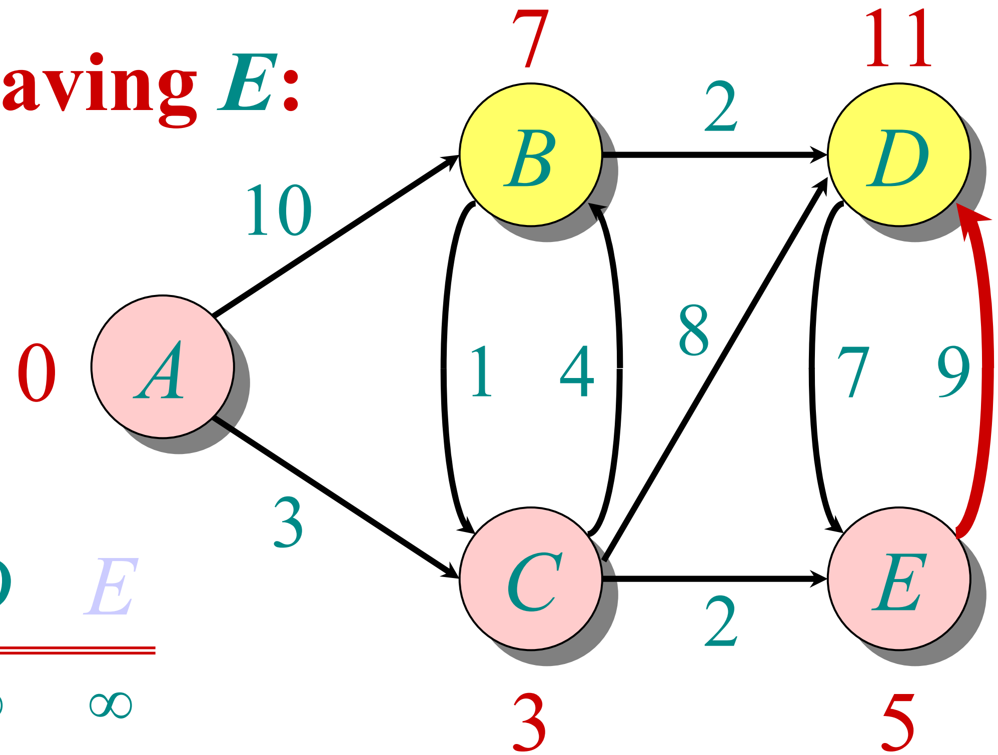
A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞
	10	3	∞	∞
	7		11	5

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



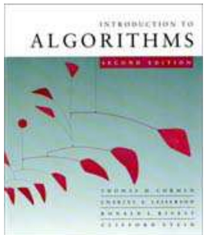
Example of Dijkstra's algorithm

Relax all edges leaving *E*:



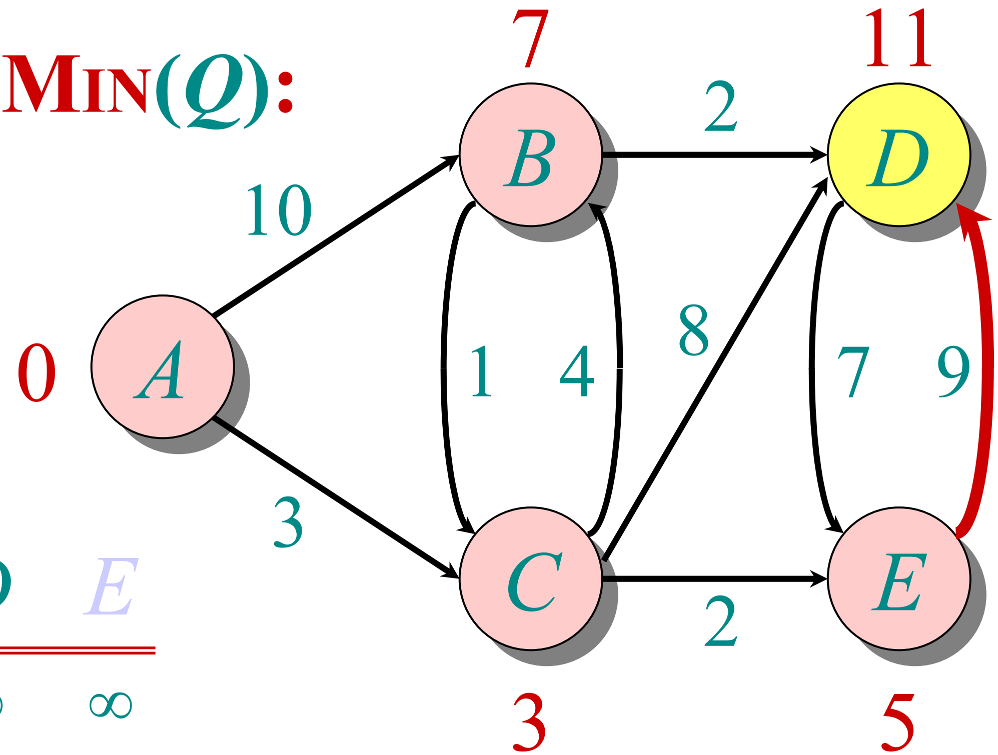
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

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



Example of Dijkstra's algorithm

"*B*" ← EXTRACT-MIN(*Q*):



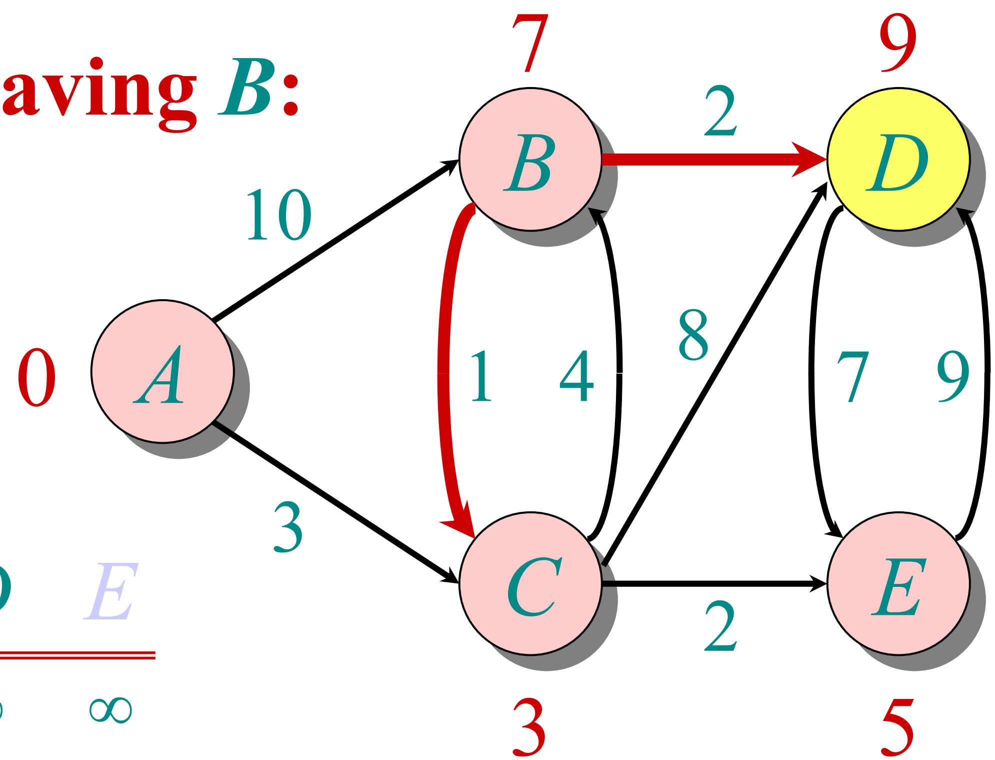
Q:

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

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

Example of Dijkstra's algorithm

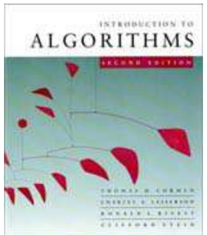
Relax all edges leaving *B*:



Q:

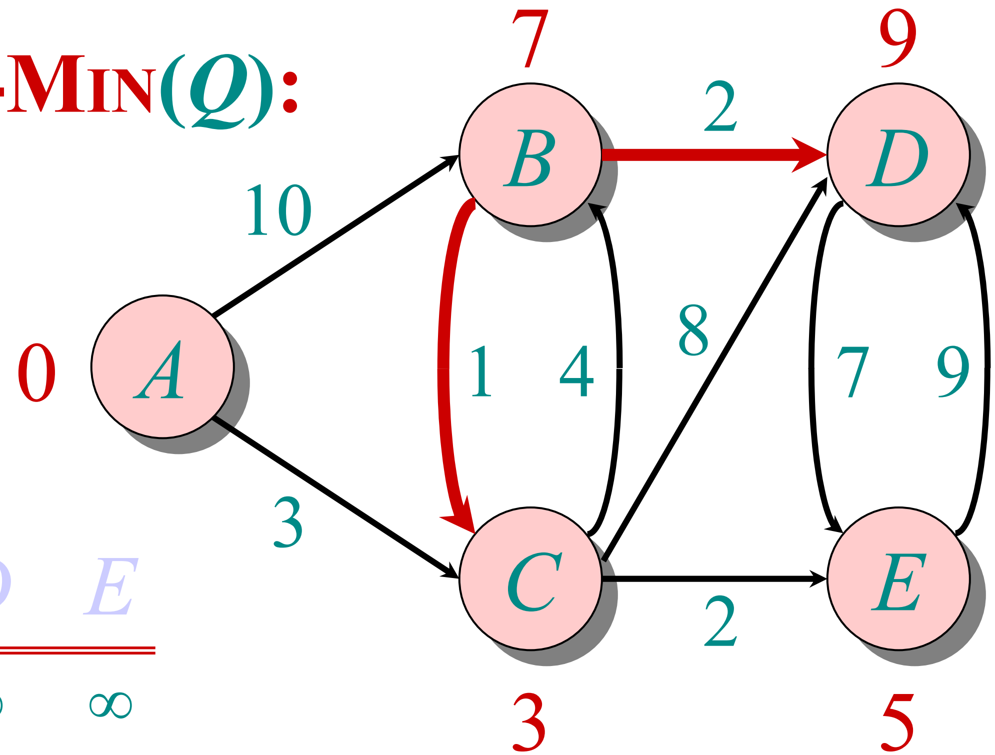
A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞
	10	3	∞	∞
	7		11	5
	7		11	
			9	

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



Example of Dijkstra's algorithm

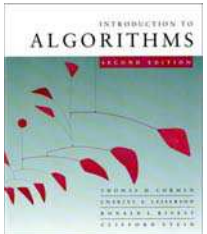
"*D*" ← EXTRACT-MIN(*Q*):



Q:

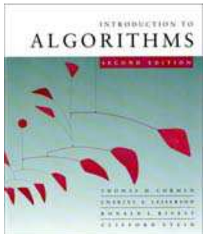
A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞
	10	3	∞	∞
	7		11	5
	7		11	
			9	

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.



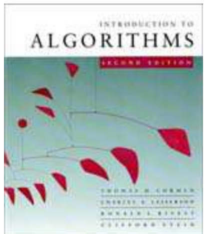
Correctness — Part I

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,

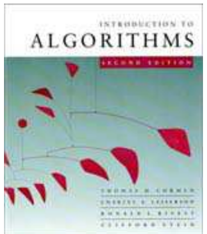
$$\begin{aligned} d[v] &< \delta(s, v) && \text{supposition} \\ &\leq \delta(s, u) + \delta(u, v) && \text{triangle inequality} \\ &\leq \delta(s,u) + w(u, v) && \text{sh. path} \leq \text{specific path} \\ &\leq d[u] + w(u, v) && v \text{ is first violation} \end{aligned}$$

Contradiction. □



Correctness — Part II

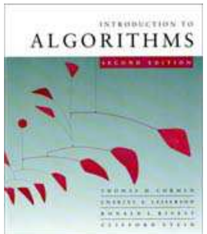
Lemma. Let u be v's predecessor on a shortest path from s to v. Then, if $d[u] = \delta(s, u)$ and edge (u, v) is relaxed, we have $d[v] = \delta(s, v)$ after the relaxation.



Correctness — Part II

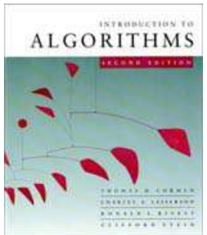
Lemma. Let u be v's predecessor on a shortest path from s to v. Then, if $d[u] = \delta(s, u)$ and edge (u, v) is relaxed, we have $d[v] = \delta(s, v)$ after the relaxation.

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

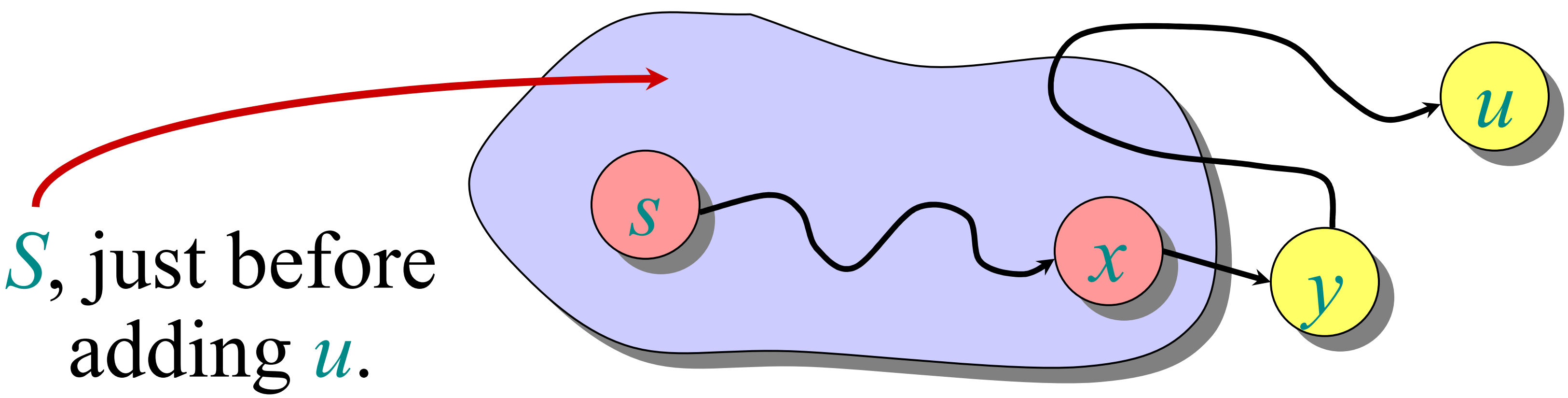
Theorem. Dijkstra's algorithm terminates with $d[v] = \delta(s, v)$ for all $v \in V$.

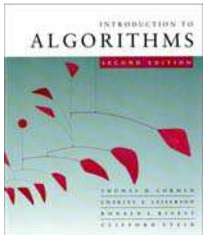


Correctness — Part III

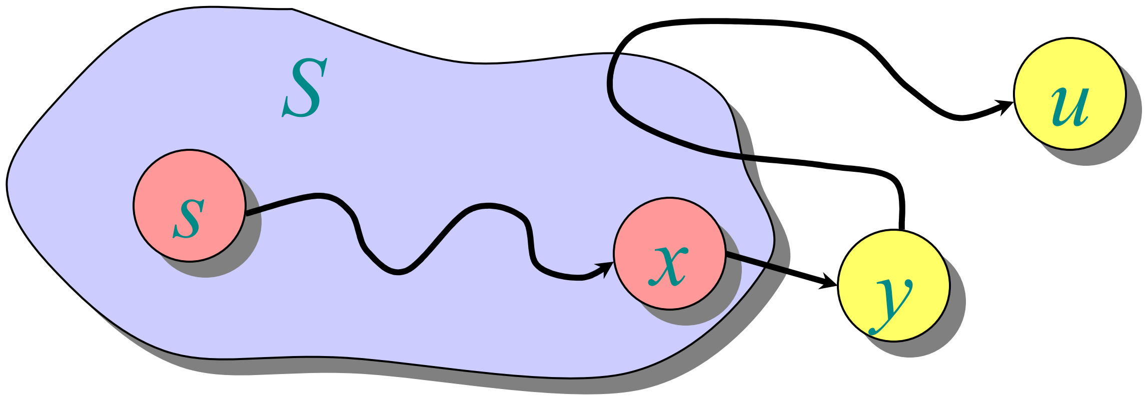
Theorem. Dijkstra's algorithm terminates with $d[v] = \delta(s, v)$ for all $v \in V$.

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:

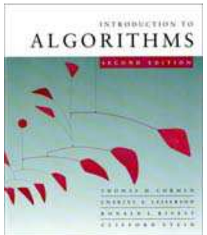




Correctness — Part III (continued)



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 $Q \neq \emptyset$

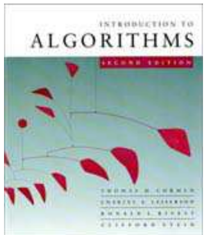
  do $u \leftarrow \text{EXTRACT-MIN}(Q)$

    $S \leftarrow S \cup \{u\}$

    for each $v \in Adj[u]$

      do if $d[v] > d[u] + w(u, v)$

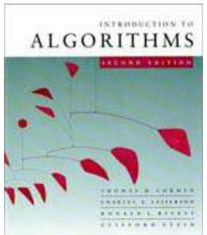
        then $d[v] \leftarrow d[u] + w(u, v)$



Analysis of Dijkstra

$|V|$ times:

while $Q \neq \emptyset$
  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 $d[v] \leftarrow d[u] + w(u, v)$



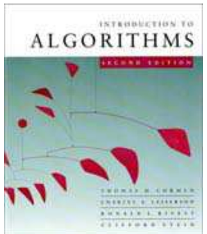
Analysis of Dijkstra

$|V|$ times:

while $Q \neq \emptyset$
  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

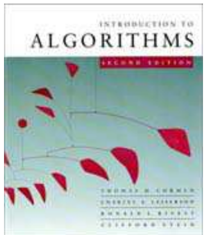
$|V|$ times:

while $Q \neq \emptyset$
 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

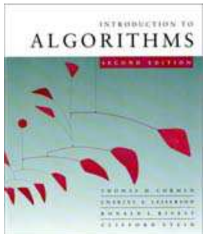
$|V|$ times:

while $Q \neq \emptyset$
  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 $= \Theta(V \cdot T_{\text{EXTRACT-MIN}} + E \cdot T_{\text{DECREASE-KEY}})$

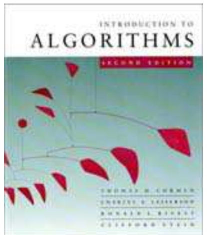
Note: Same formula as in the analysis of Prim's minimum spanning tree algorithm.



Analysis of Dijkstra (continued)

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

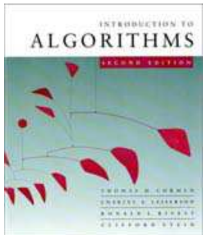
Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total



Analysis of Dijkstra (continued)

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

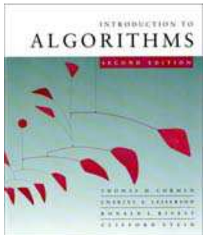
Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$



Analysis of Dijkstra (continued)

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

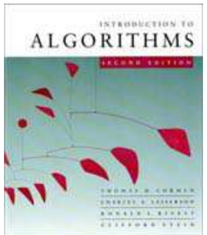
Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$



Analysis of Dijkstra (continued)

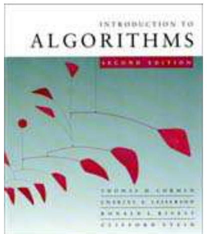
Time = $\Theta(V) \cdot T_{\text{EXTRACT-MIN}} + \Theta(E) \cdot T_{\text{DECREASE-KEY}}$

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

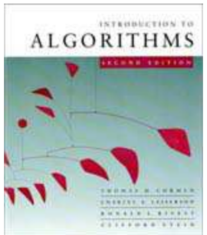
Suppose that $w(u, v) = 1$ for all $(u, v) \in E$. Can Dijkstra's algorithm be improved?



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.



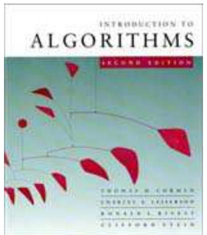
Unweighted graphs

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 $Q \neq \emptyset$
    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?

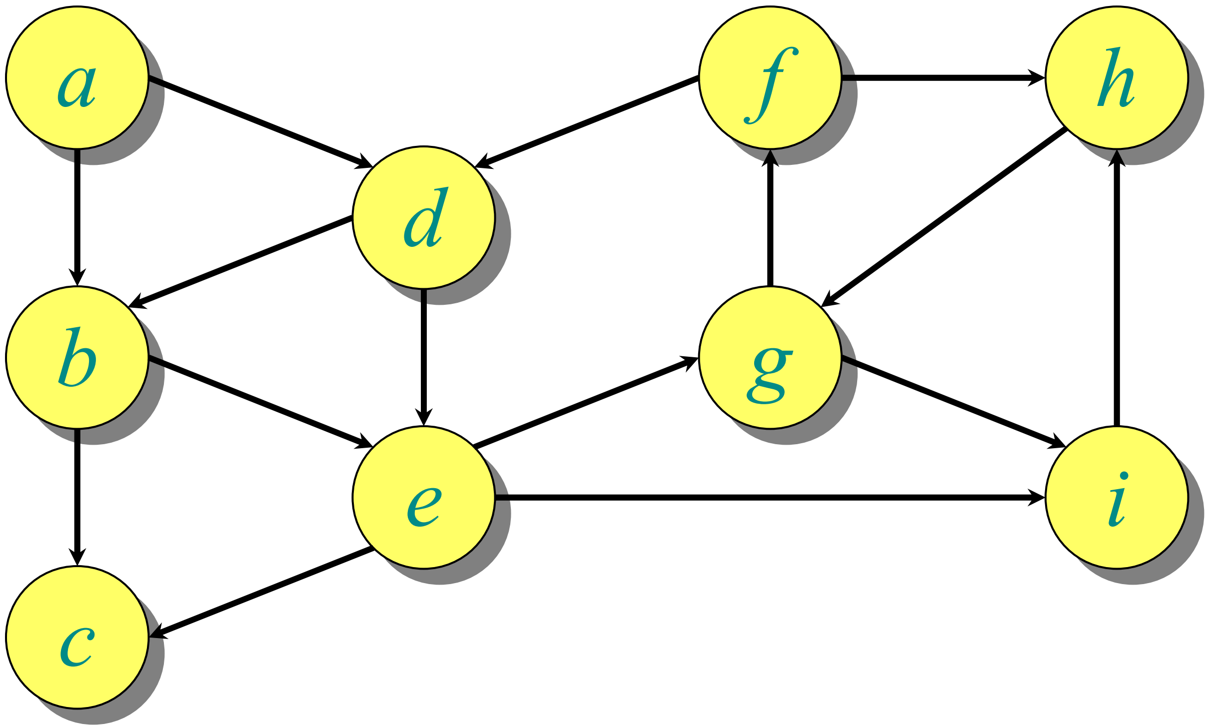
- Use a simple FIFO queue instead of a priority queue.

Breadth-first search

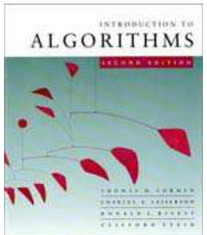
while $Q \neq \emptyset$
  do $u \leftarrow$ DEQUEUE(Q)
    for each $v \in Adj[u]$
      do if $d[v] = \infty$
        then $d[v] \leftarrow d[u] + 1$
          ENQUEUE(Q, v)

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

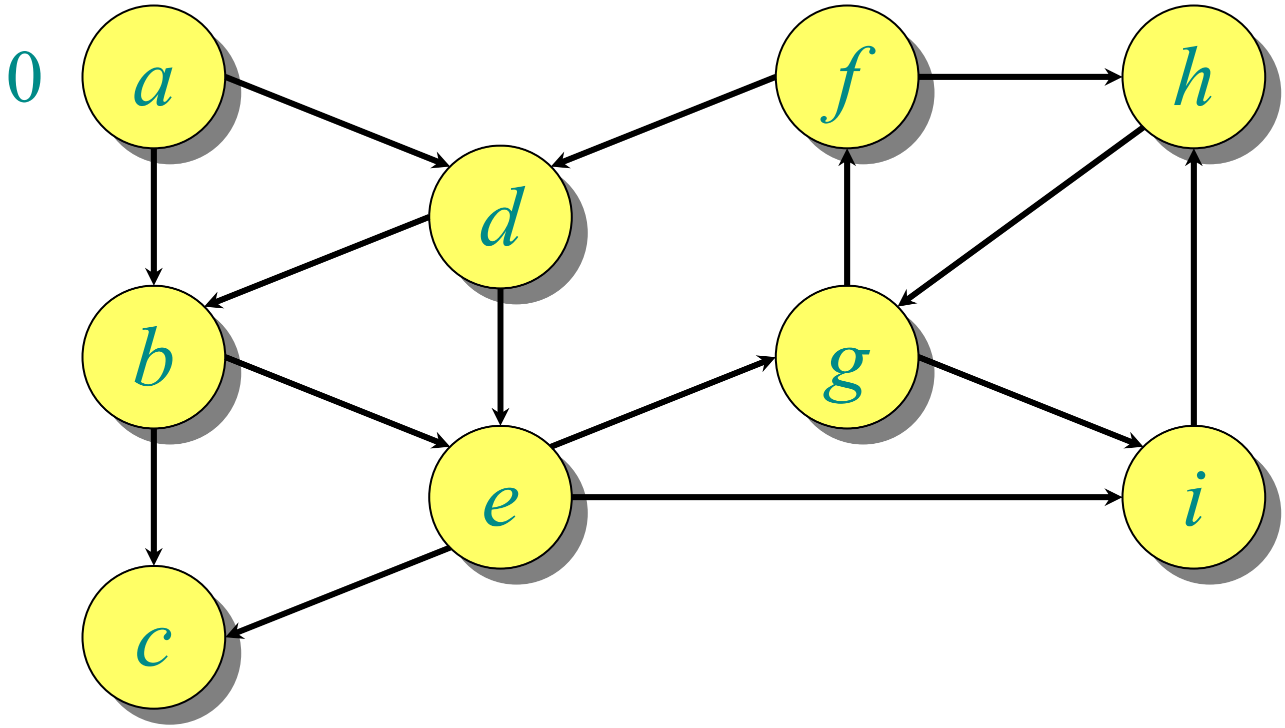
Example of breadth-first search



Q:



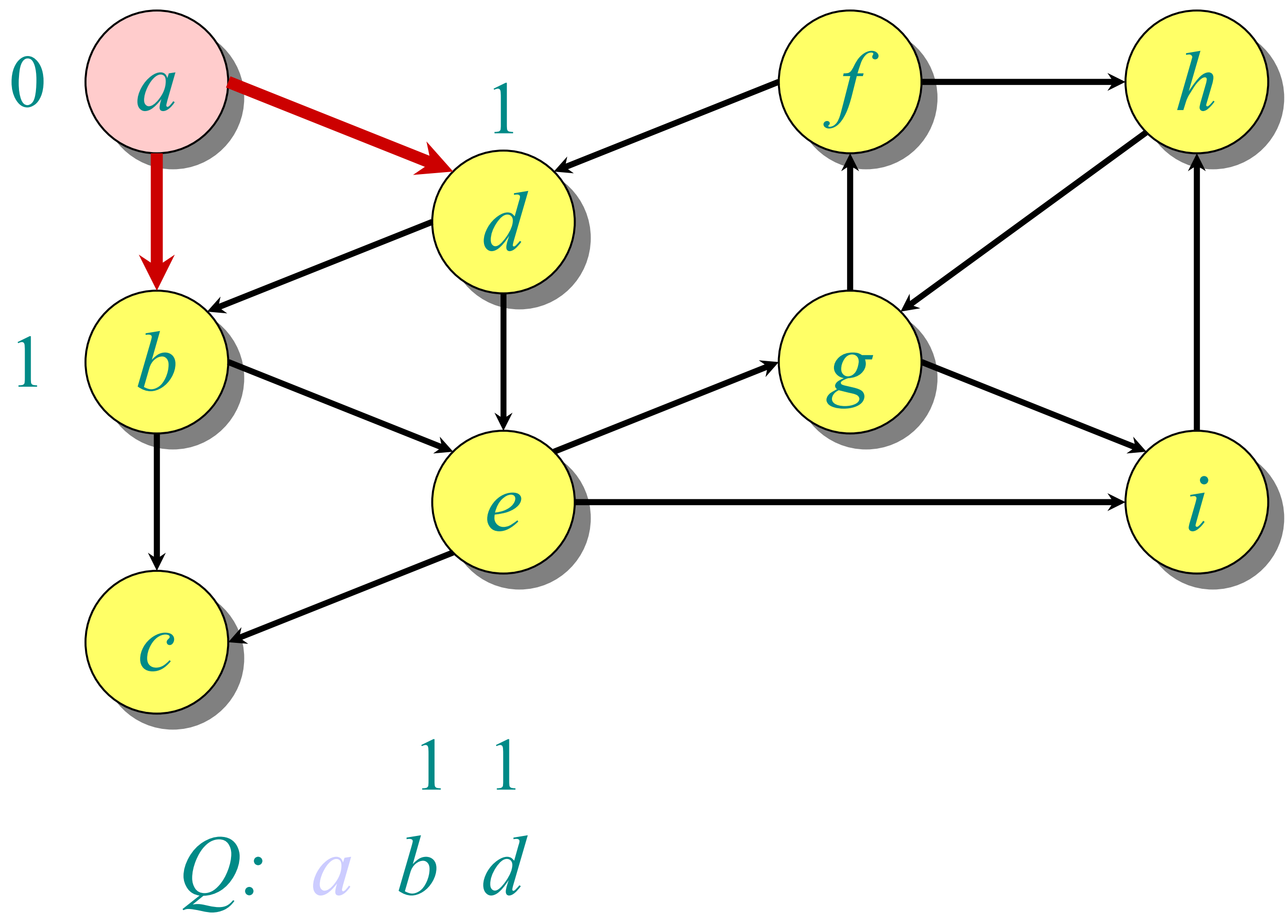
Example of breadth-first search



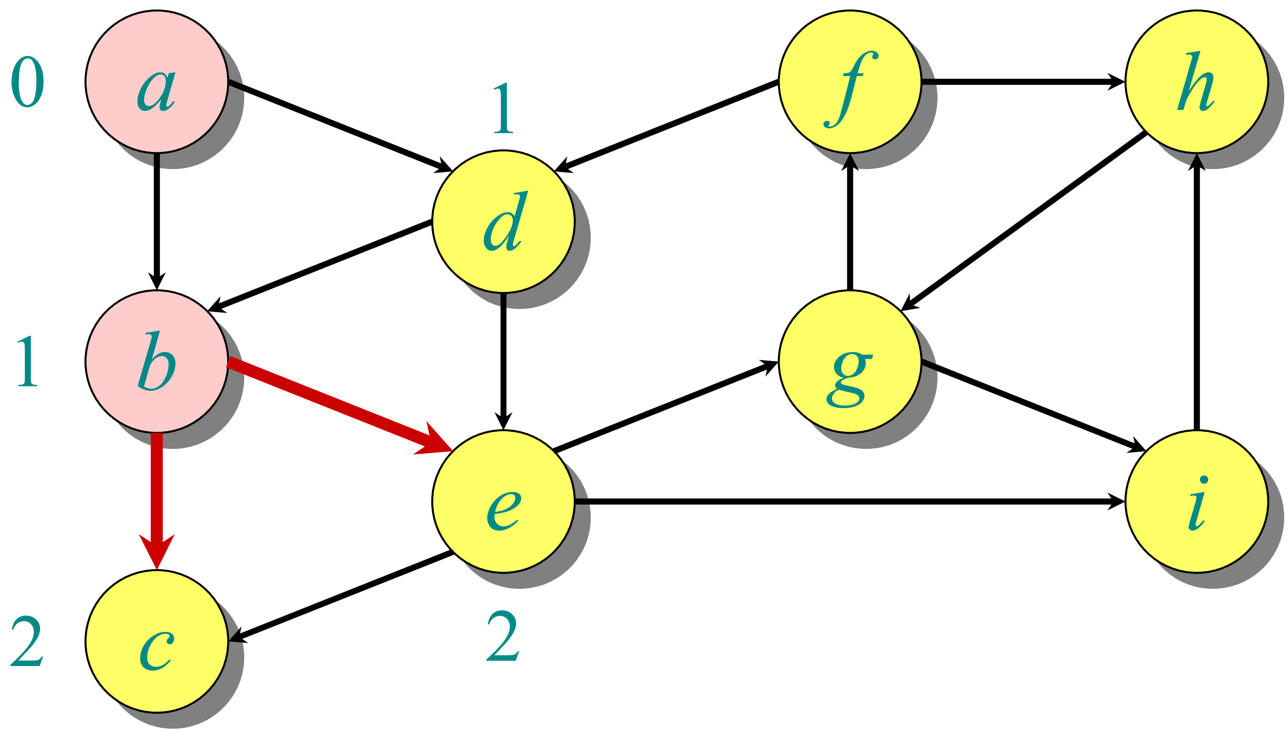
0

Q: *a*

Example of breadth-first search



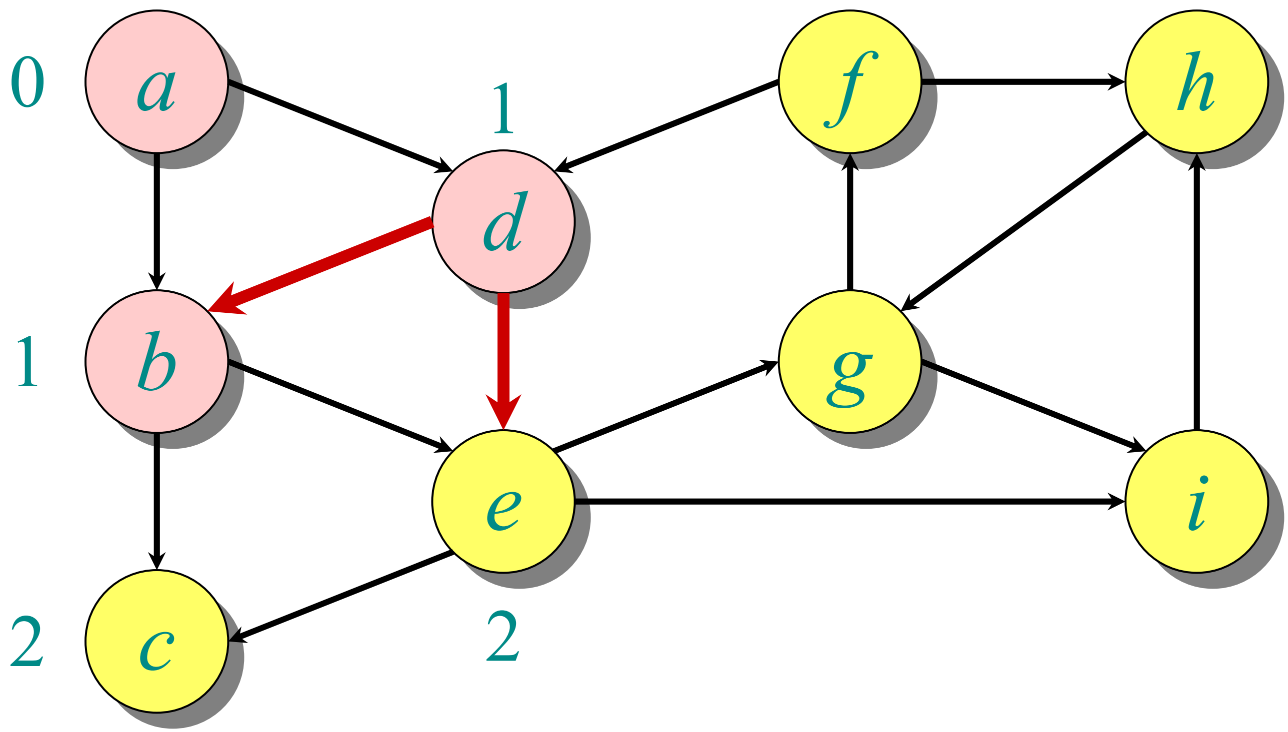
Example of breadth-first search



1 2 2

Q: a b d c e

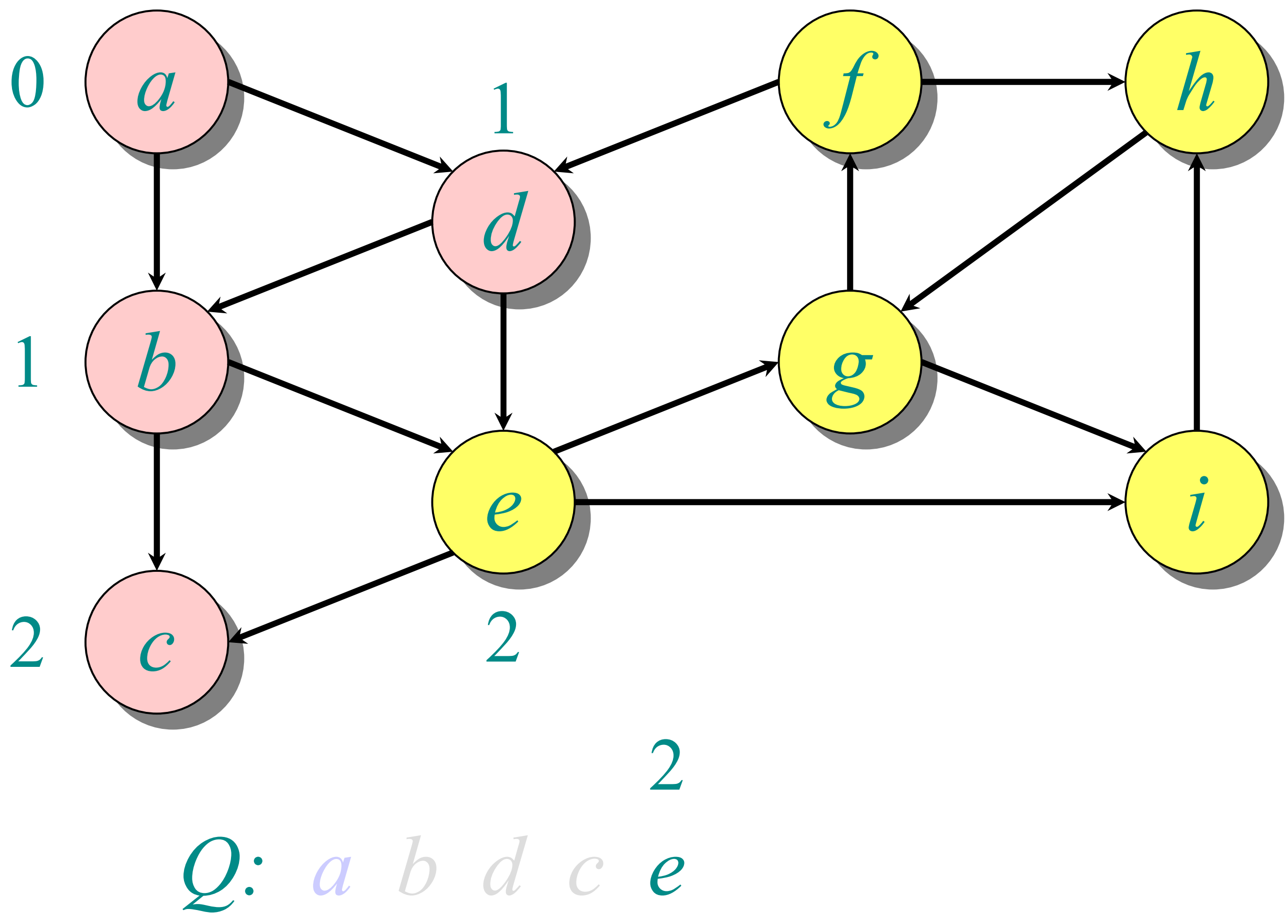
Example of breadth-first search



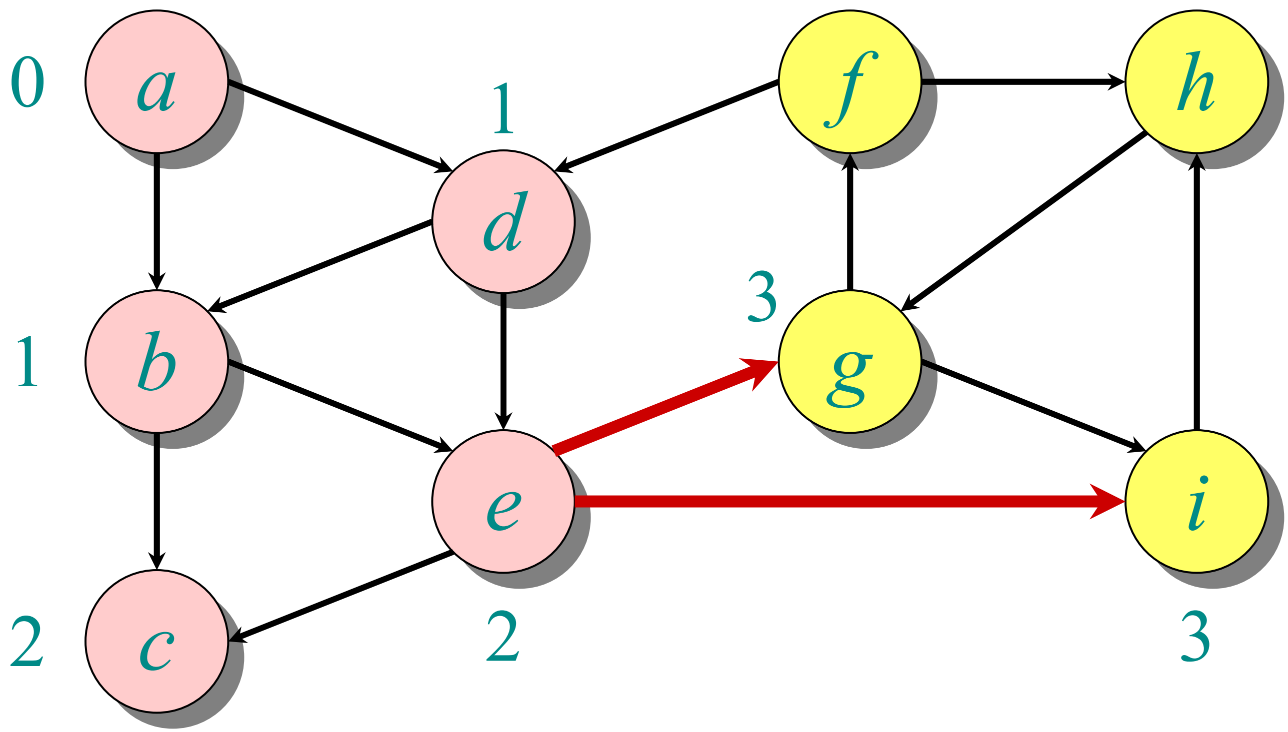
2 2

Q: *a b d c e*

Example of breadth-first search



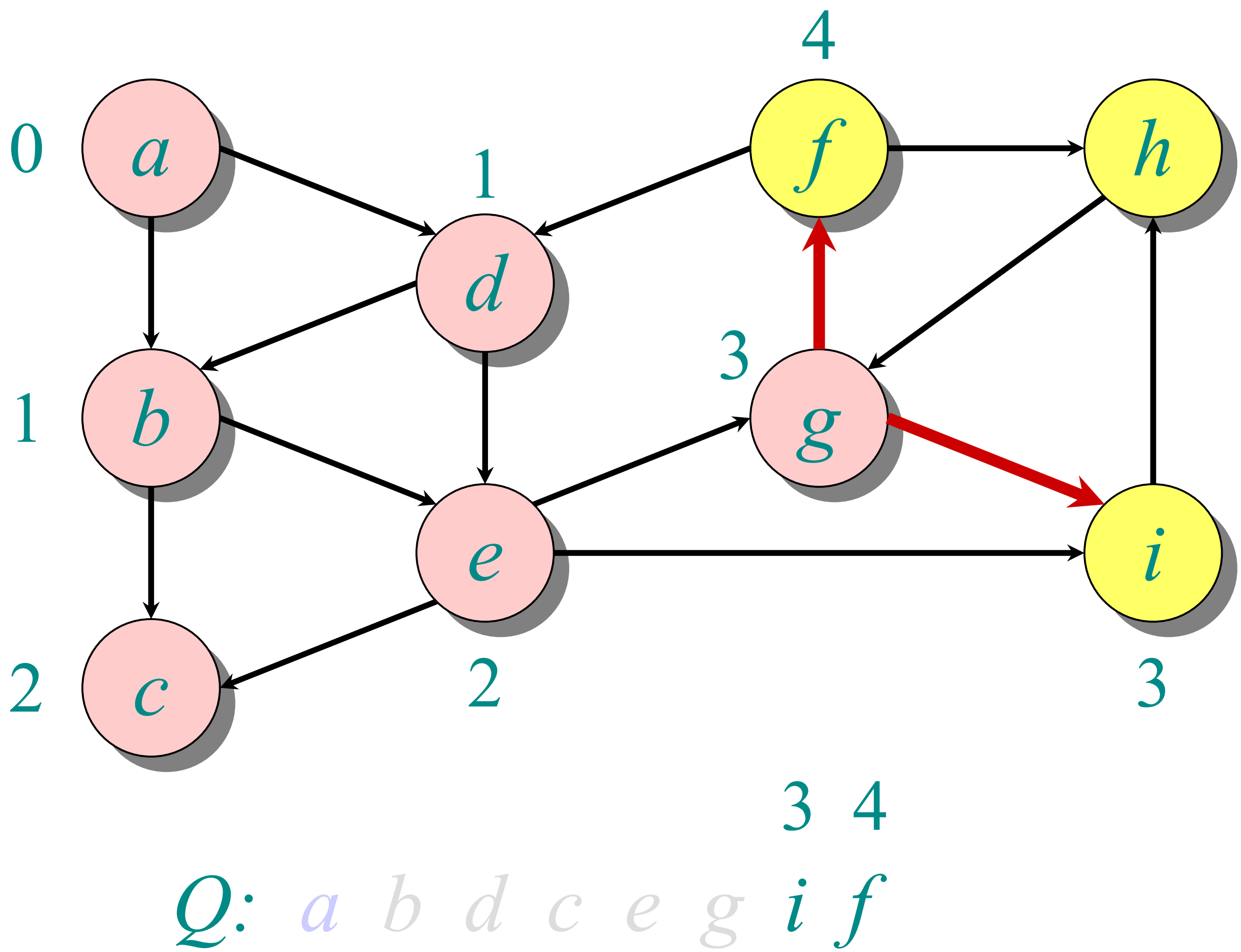
Example of breadth-first search



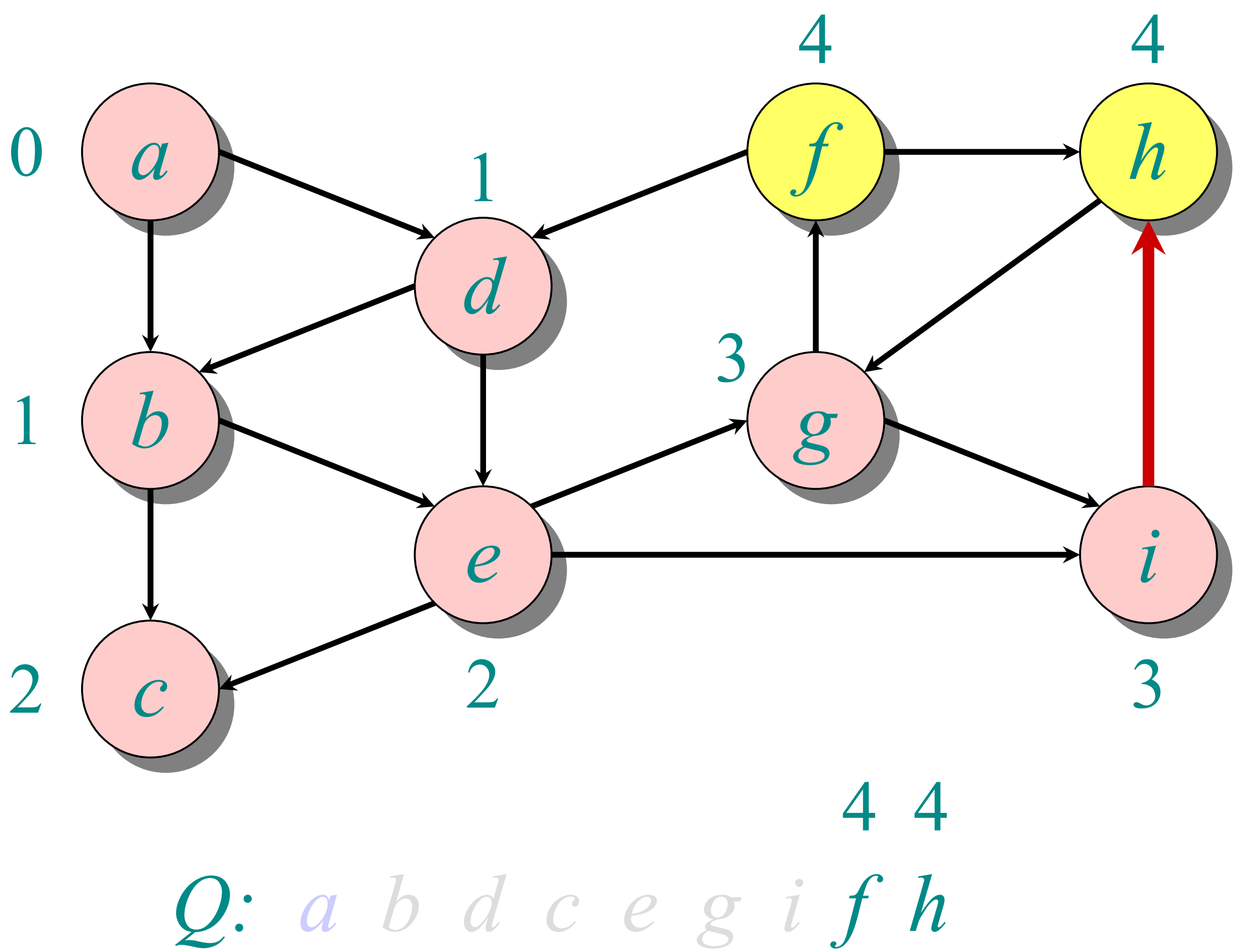
3 3

Q: a b d c e g i

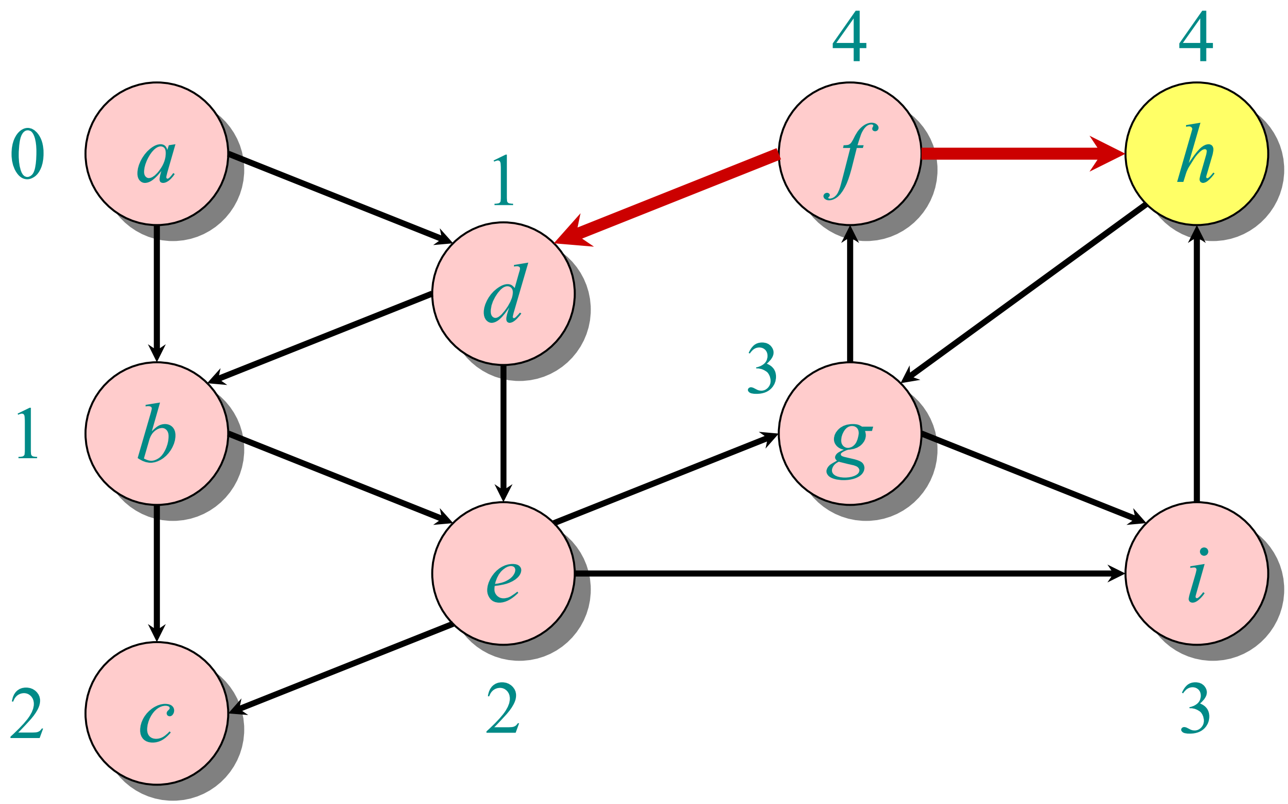
Example of breadth-first search



Example of breadth-first search

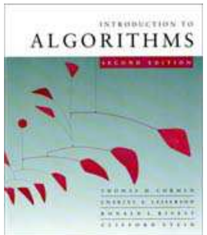


Example of breadth-first search

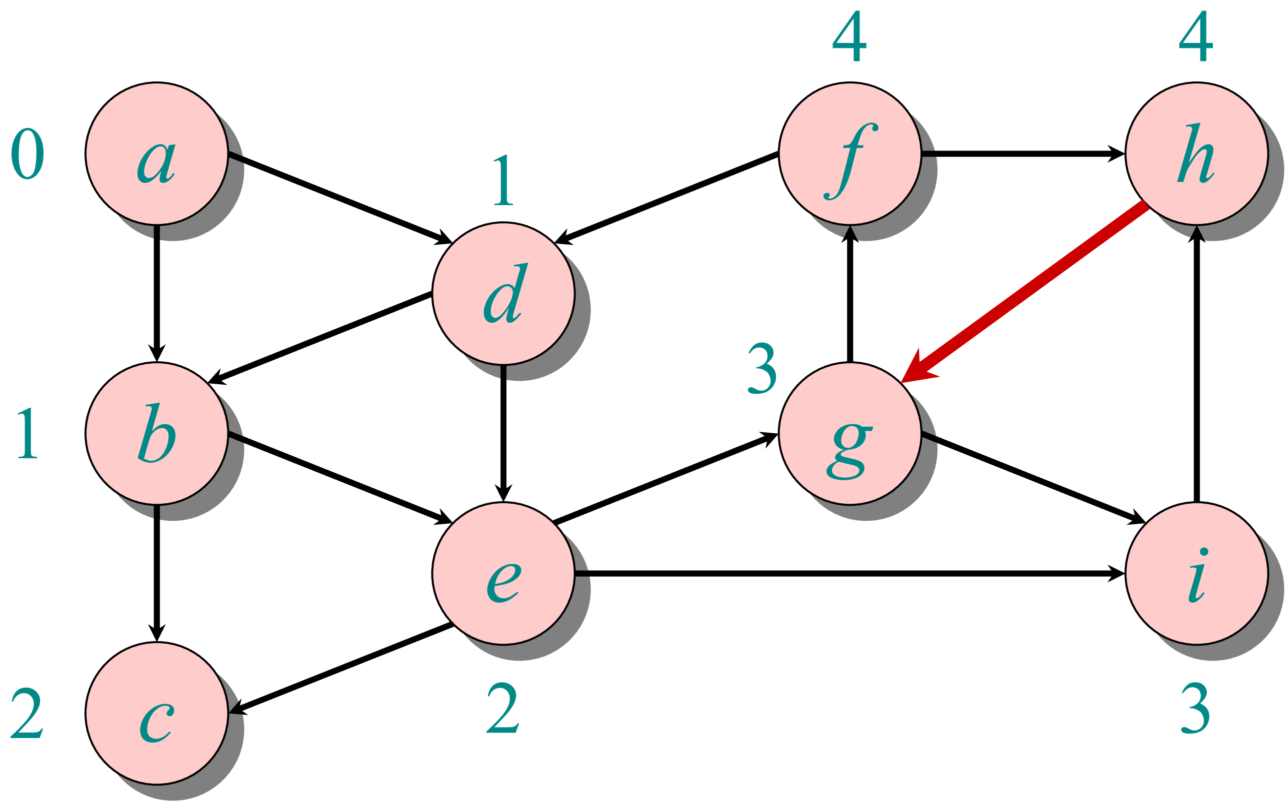


4

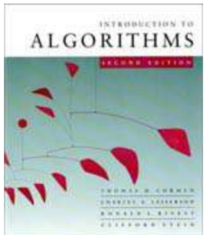
Q: *a b d c e g i f h*



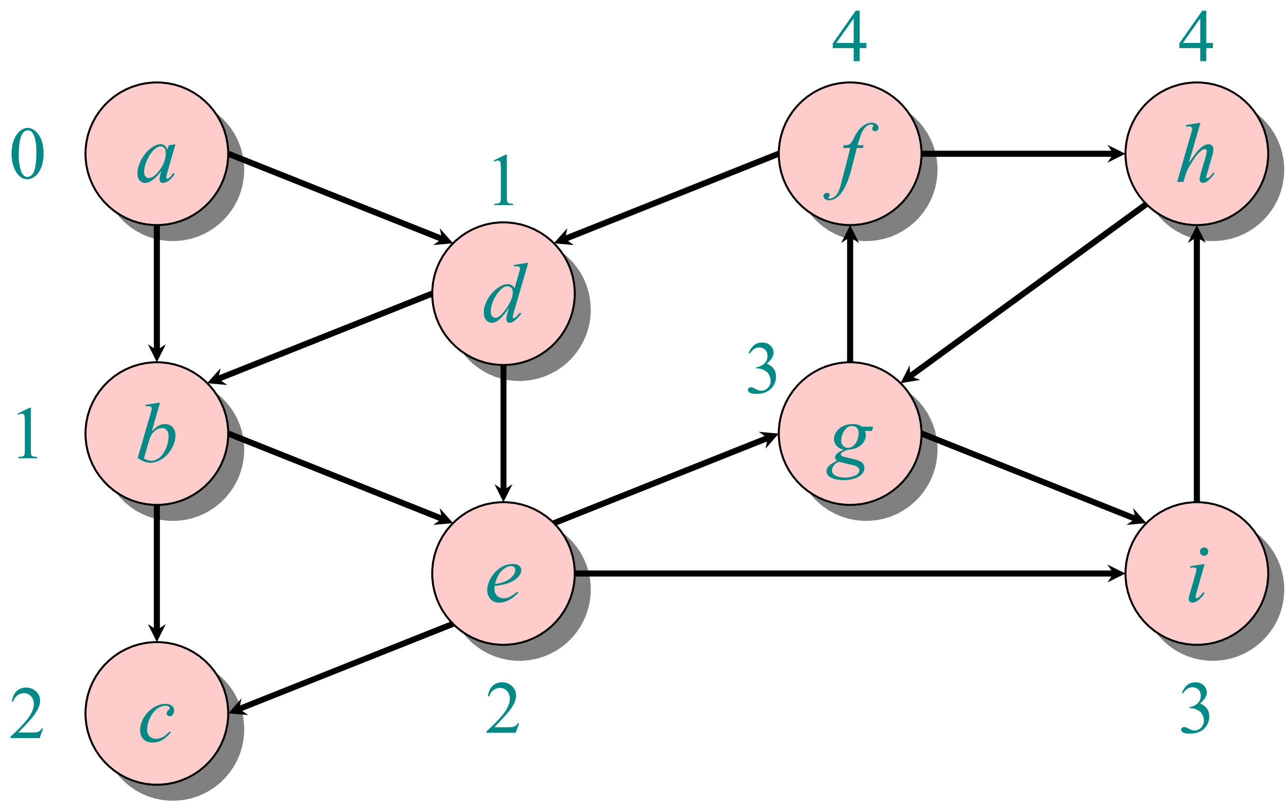
Example of breadth-first search



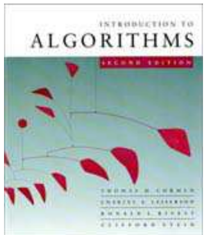
Q: *a b d c e g i f h*



Example of breadth-first search



Q: *a b d c e g i f h*



Correctness of BFS

```
while Q ≠ ∅
    do u ← DEQUEUE(Q)
        for each v ∈ Adj[u]
            do if d[v] = ∞
                then d[v] ← 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 $d[v] = d[u] + 1$.

