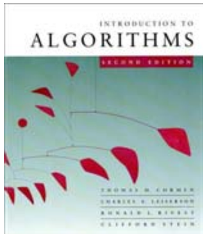
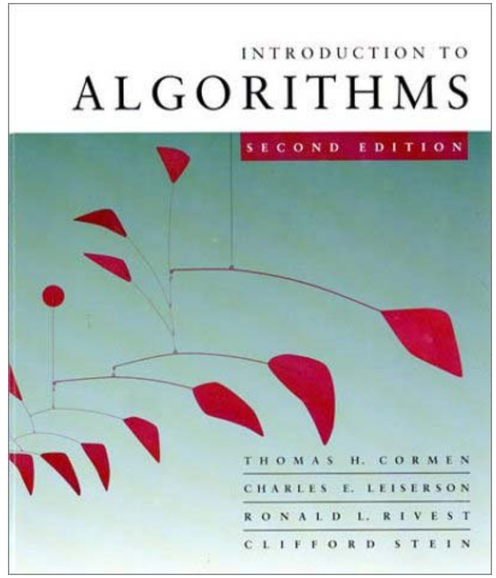


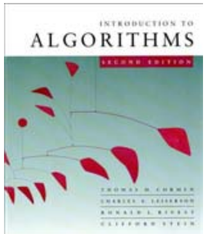

# CS 5633 -- Spring 2004




# *Single Source Shortest Paths*

**Carola Wenk**

Slides courtesy of Charles Leiserson with small changes by Carola Wenk

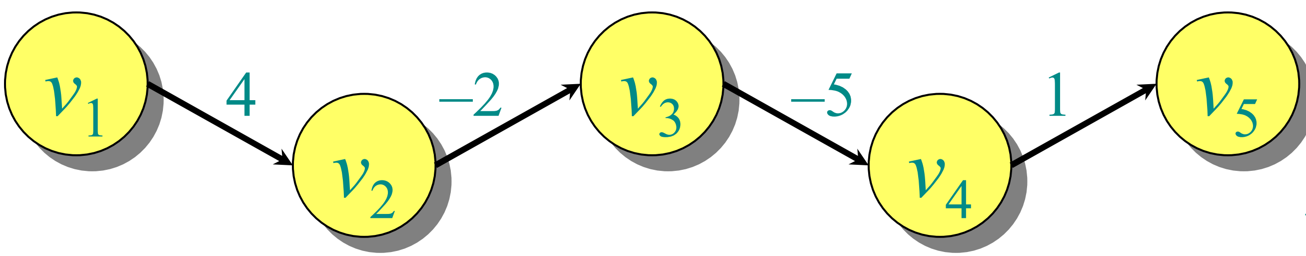



# Paths in graphs

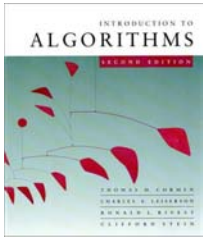
Consider a digraph $G = (V, E)$ with edge-weight function $w : E \to \mathbb{R}$. The ***weight*** of path $p = v_1 \to v_2 \to \ldots \to v_k$ is defined to be

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

**Example:**




$w(p) = -2$

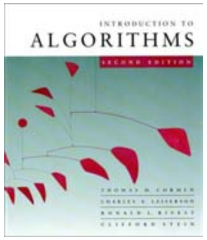



# Shortest paths

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

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

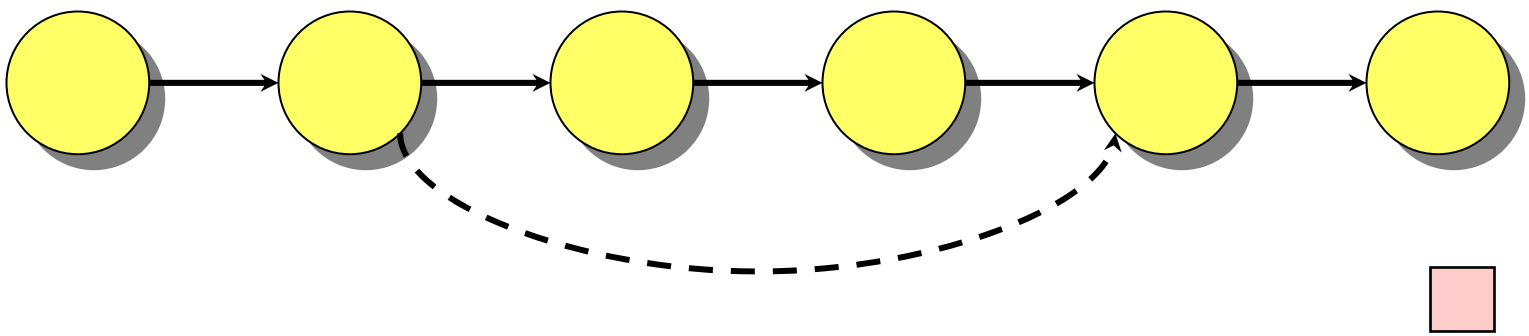
**Note:** $\delta(u, v) = \infty$ if no path from $u$ to $v$ exists.




# Optimal substructure

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

*Proof.* Cut and paste:

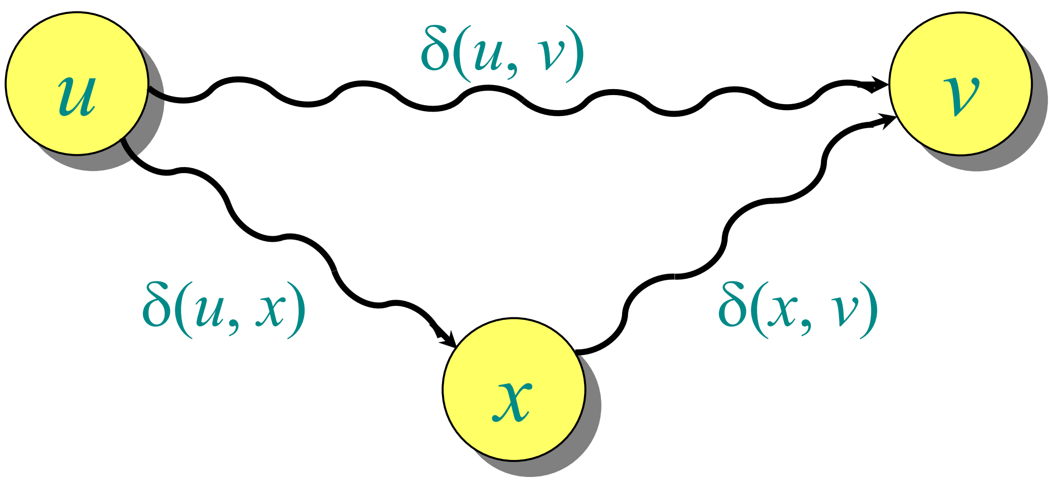


# Triangle inequality

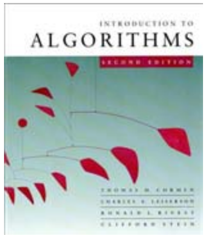
**Theorem.** For all $u, v, x \in V$, we have

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

*Proof.*



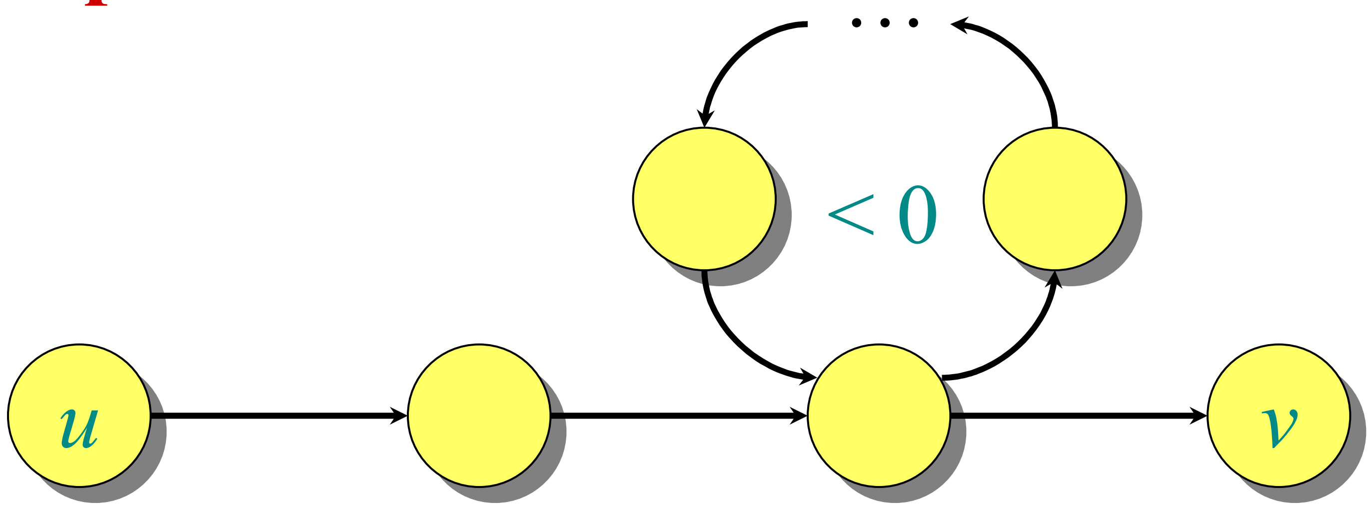

∎

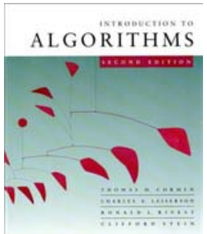



# Well-definedness of shortest paths

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

**Example:**





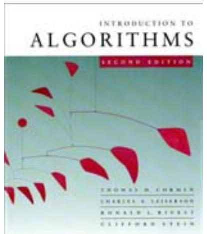

# Single-source shortest paths

**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 weights 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 $v$.

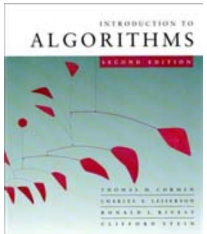



# Dijkstra's algorithm

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




# Dijkstra

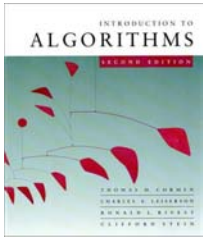
$d[s] \leftarrow 0$
**for** each $v \in V - \{s\}$
    **do** $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$    ▷ $Q$ is
**while** $Q \neq \varnothing$
    **do** $u \leftarrow$ EXTRACT-MIN$(Q)$
        $S \leftarrow S \cup \{u\}$
        **for** each $v \in Adj[u]$
           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

***relaxation step***

Implicit DECREASE-KEY

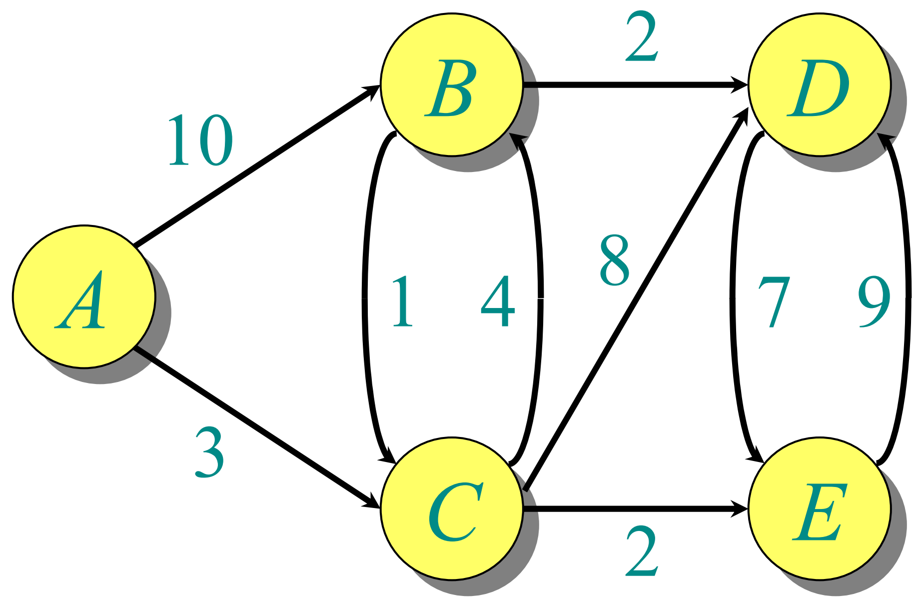
**PRIM's algorithm**

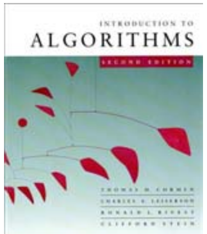
$Q \leftarrow V$
$key[v] \leftarrow \infty$ for all $v \in V$
$key[s] \leftarrow 0$ for some arbitrary $s \in V$
**while** $Q \neq \varnothing$
    **do** $u \leftarrow$ EXTRACT-MIN$(Q)$
        **for** each $v \in Adj[u]$
            **do if** $v \in Q$ and $w(u, v) < key[v]$
                **then**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




# Example of Dijkstra’s algorithm

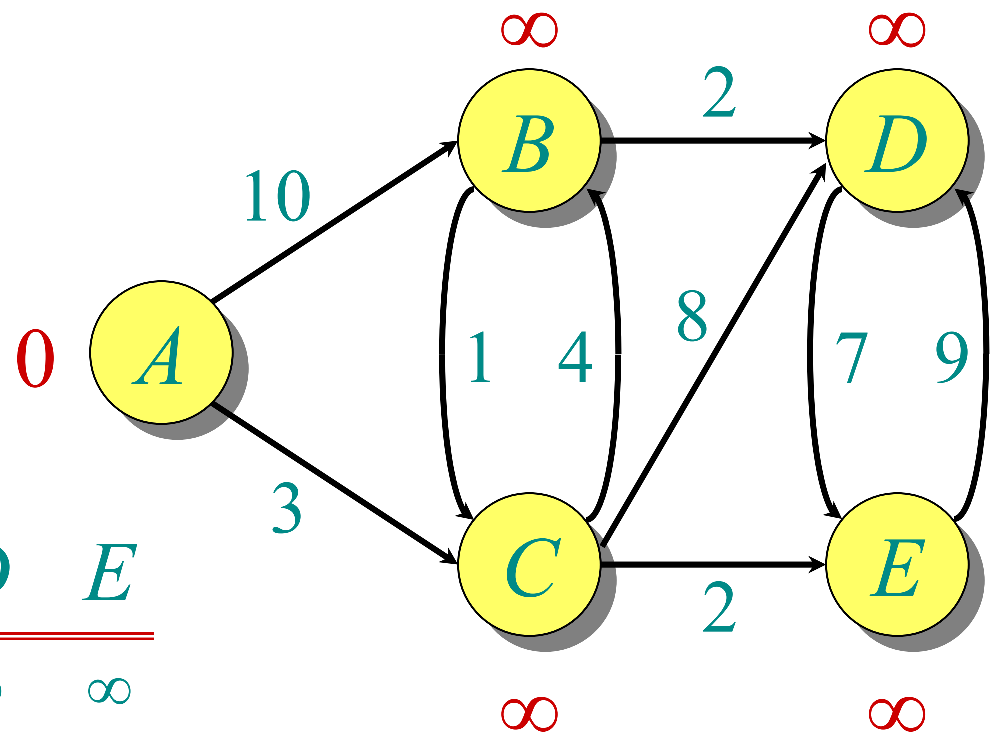
**Graph with nonnegative edge weights:**





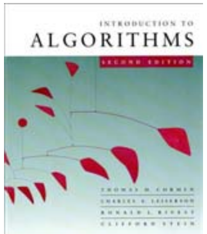

# Example of Dijkstra's algorithm

**Initialize:**



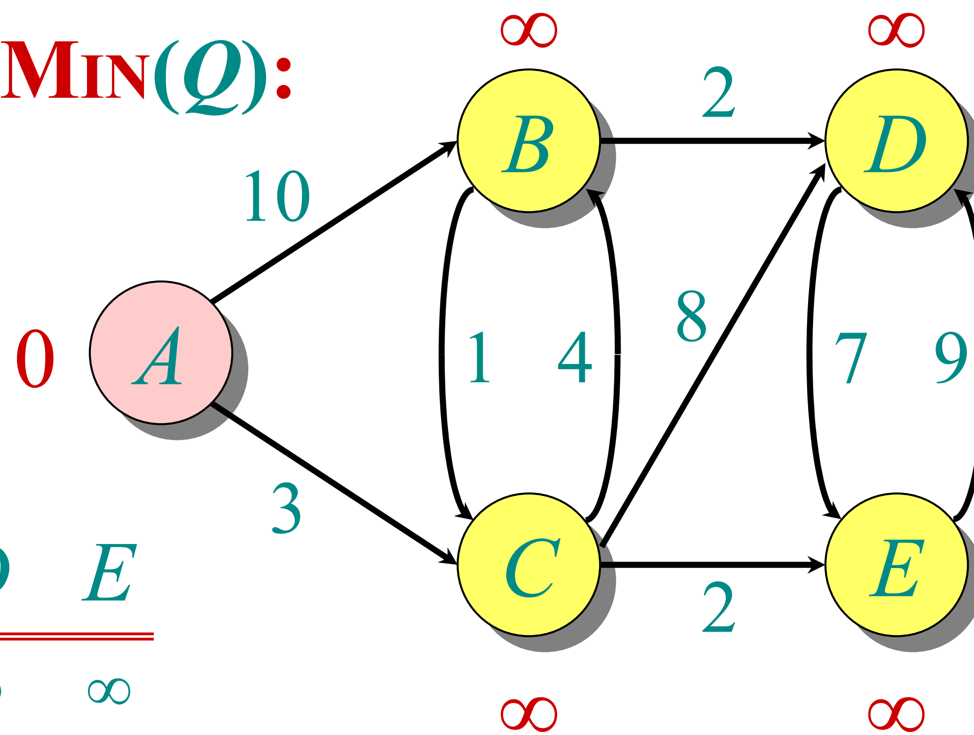

| *Q*: | *A* | *B* | *C* | *D* | *E* |
|---|---|---|---|---|---|
| | 0 | ∞ | ∞ | ∞ | ∞ |

*S*: {}



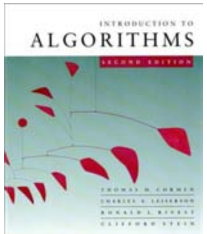

# Example of Dijkstra's algorithm

"$A$" ← **EXTRACT-MIN**($Q$):



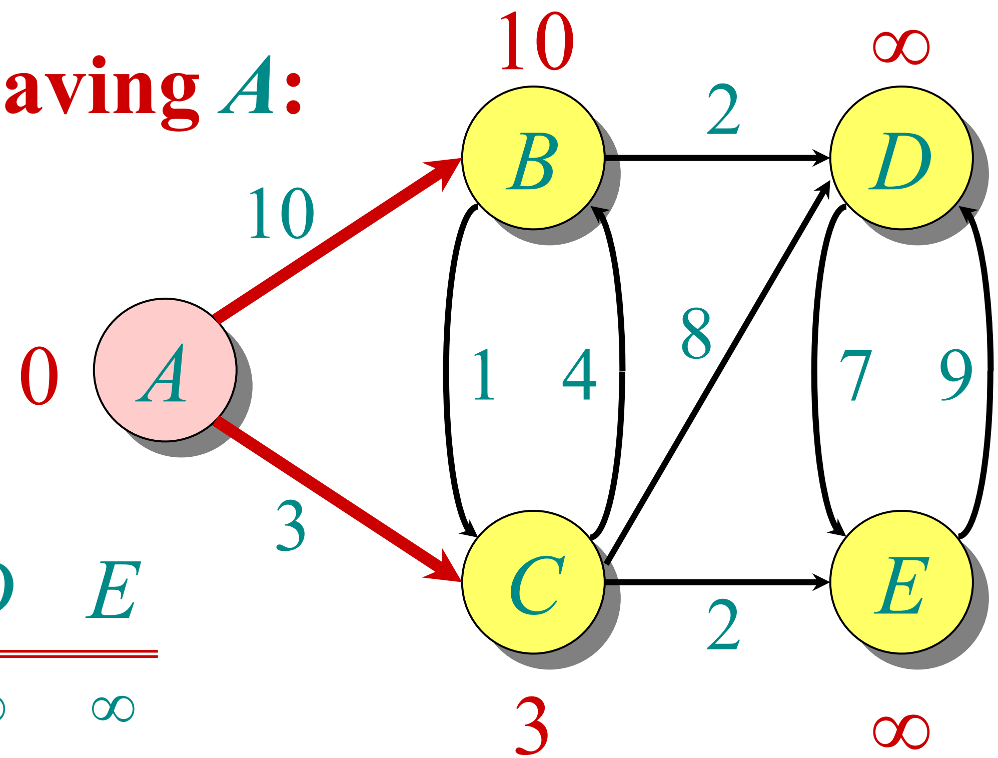

| $Q$: | $A$ | $B$ | $C$ | $D$ | $E$ |
|---|---|---|---|---|---|
| | 0 | ∞ | ∞ | ∞ | ∞ |

$S$: { $A$ }



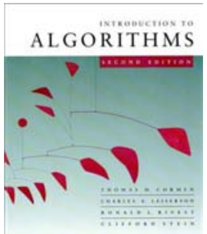

# Example of Dijkstra's algorithm

**Relax all edges leaving *A*:**



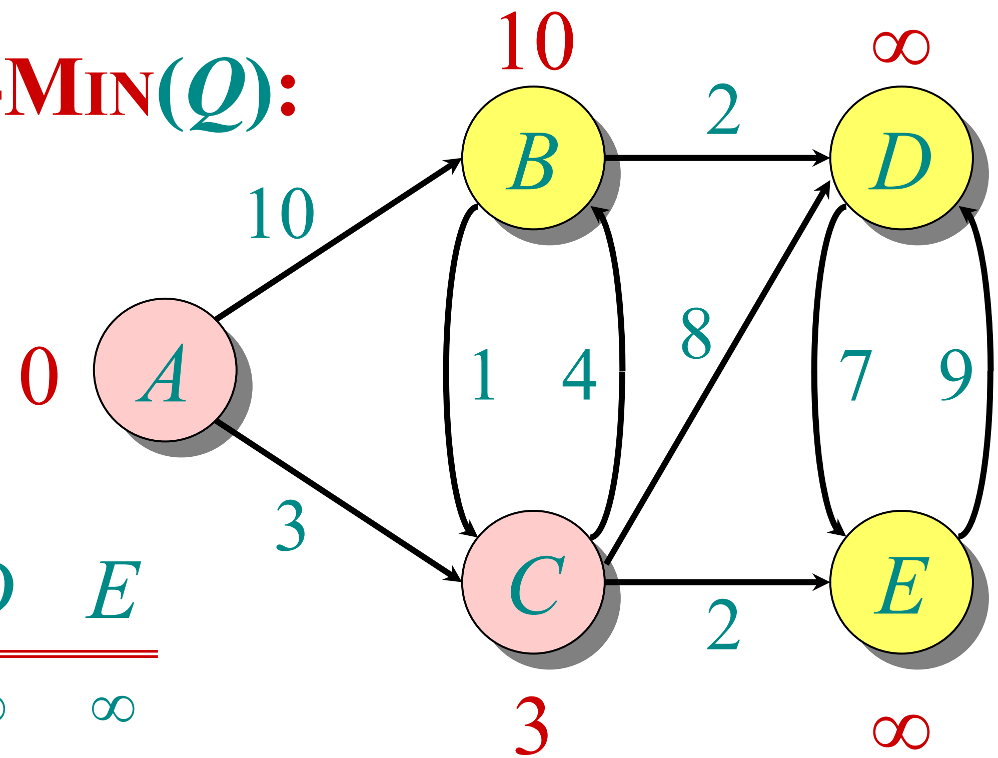

| *Q:* | *A* | *B* | *C* | *D* | *E* |
|---|---|---|---|---|---|
| | 0 | ∞ | ∞ | ∞ | ∞ |
| | | 10 | 3 | – | – |

*S*: { *A* }



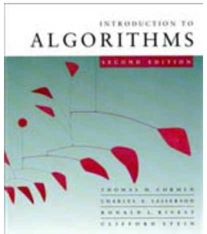

# Example of Dijkstra's algorithm

"*C*" ← EXTRACT-MIN(*Q*):



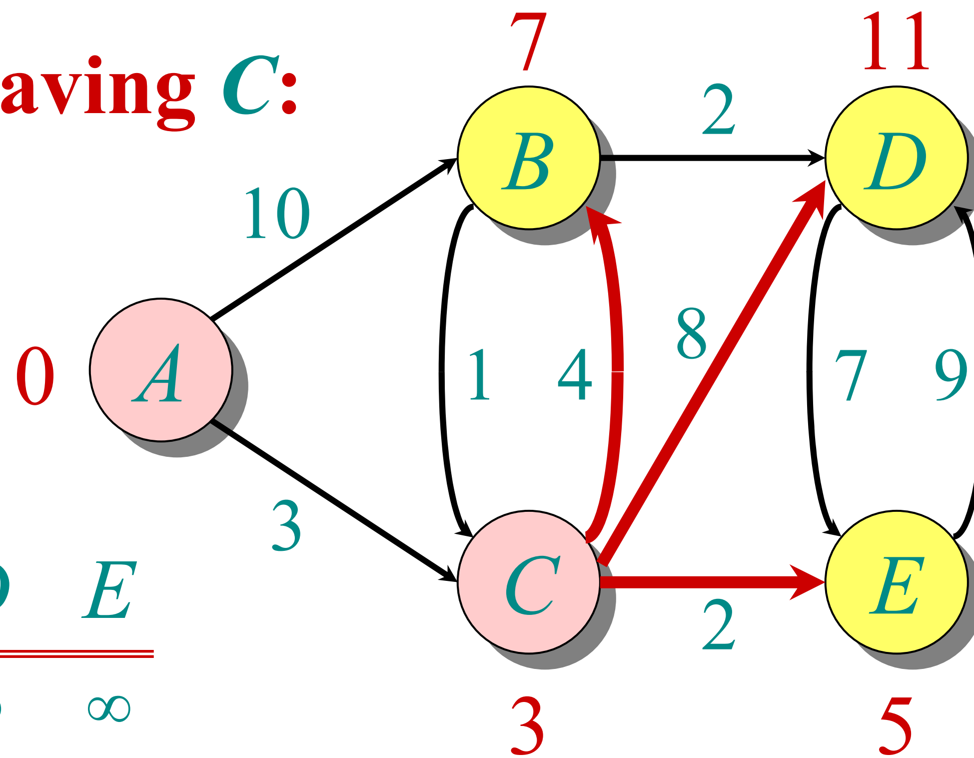

| *Q*: | *A* | *B* | *C* | *D* | *E* |
|---|---|---|---|---|---|
| | 0 | ∞ | ∞ | ∞ | ∞ |
| | | 10 | 3 | – | – |

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



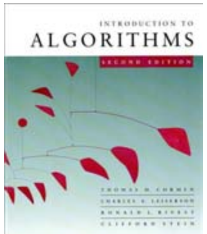

# Example of Dijkstra's algorithm

**Relax all edges leaving *C*:**



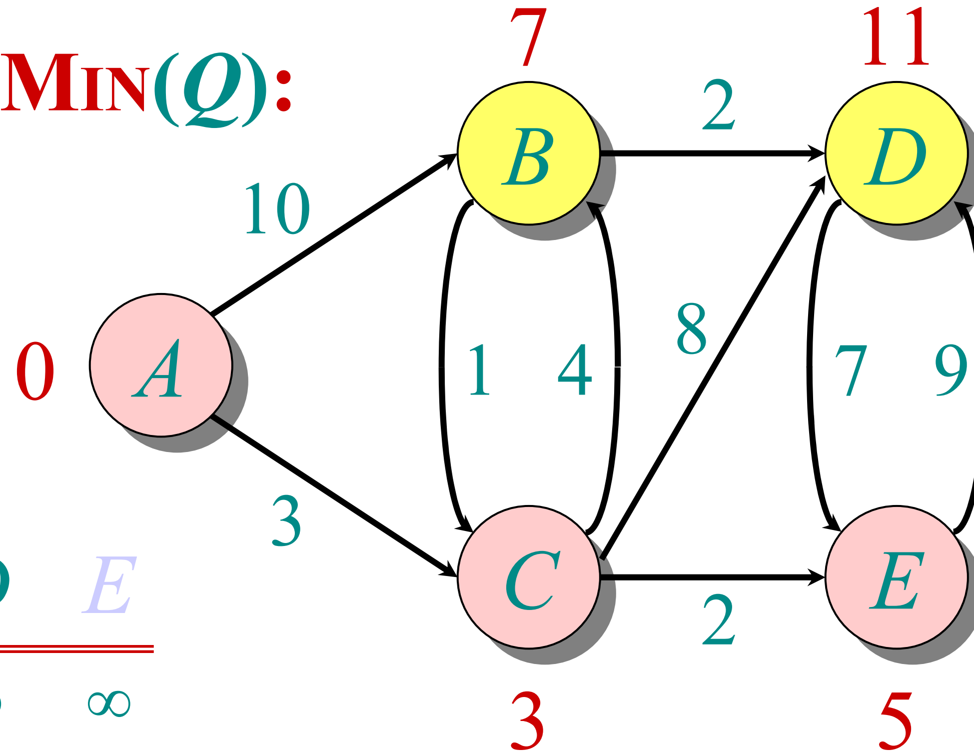

| *Q*: | *A* | *B* | *C* | *D* | *E* |
|---|---|---|---|---|---|
| | 0 | ∞ | ∞ | ∞ | ∞ |
| | | 10 | 3 | – | – |
| | | 7 | | 11 | 5 |

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



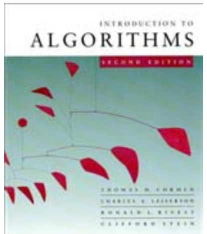

# Example of Dijkstra's algorithm

"*E*" ← EXTRACT-MIN(*Q*):



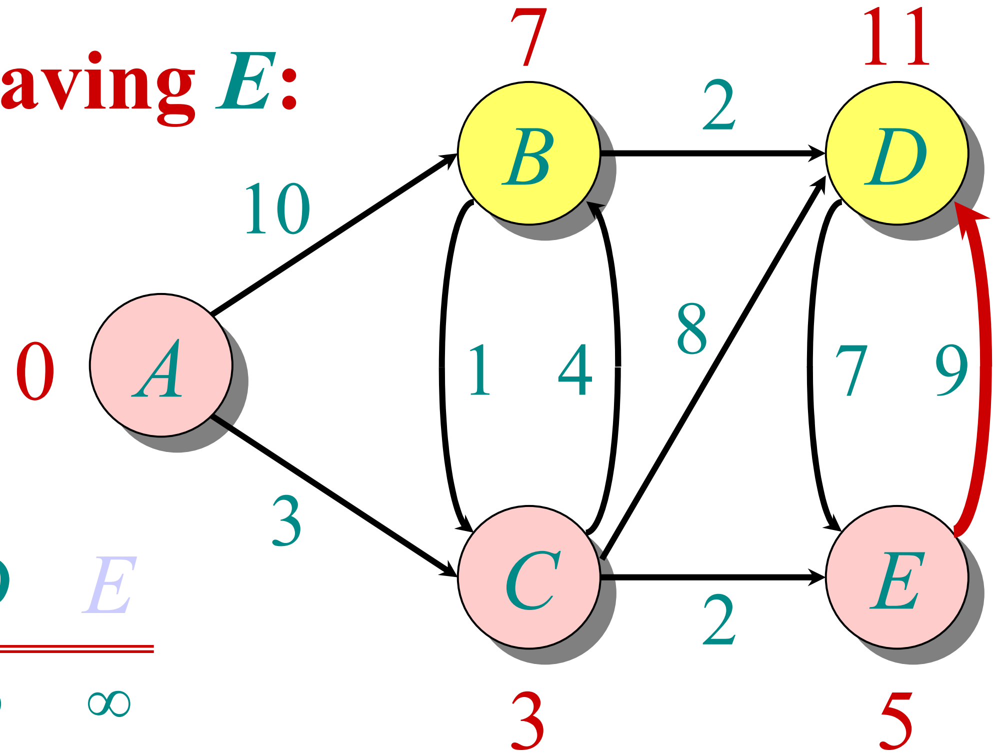

| *Q*: | *A* | *B* | *C* | *D* | *E* |
|---|---|---|---|---|---|
| | 0 | ∞ | ∞ | ∞ | ∞ |
| | | 10 | 3 | – | – |
| | | 7 | | 11 | 5 |

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



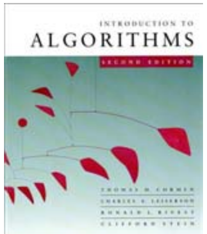

# Example of Dijkstra's algorithm

**Relax all edges leaving *E*:**



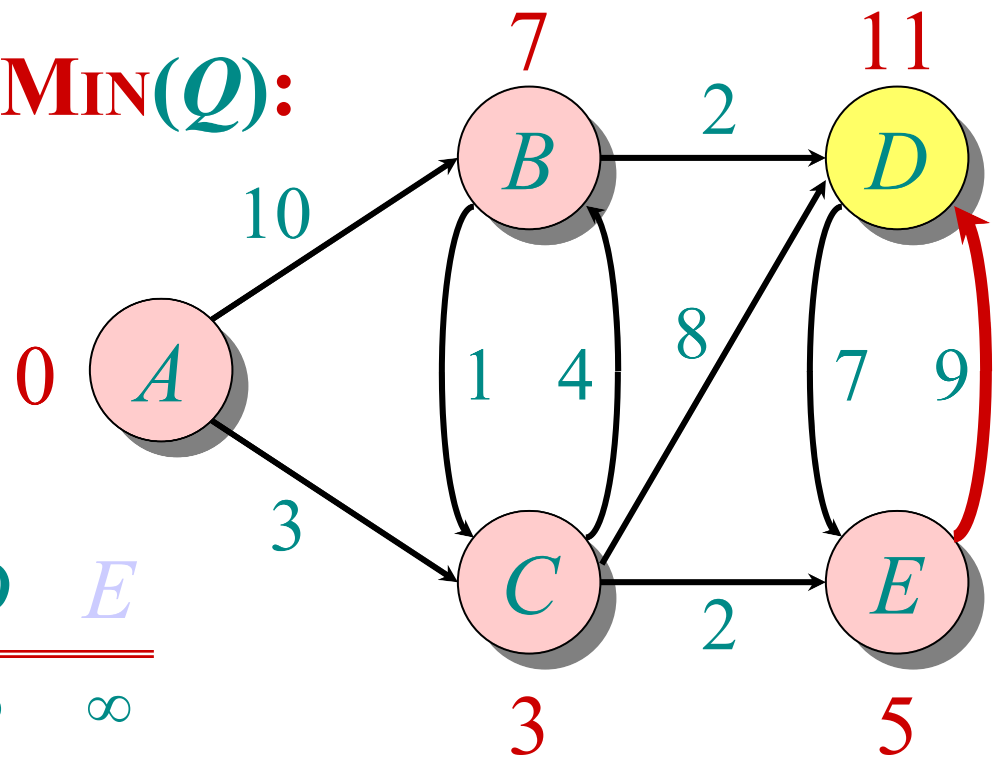

| *Q*: | *A* | *B* | *C* | *D* | *E* |
|---|---|---|---|---|---|
| | 0 | ∞ | ∞ | ∞ | ∞ |
| | | 10 | 3 | ∞ | ∞ |
| | | 7 | | 11 | 5 |
| | | 7 | | 11 | |

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



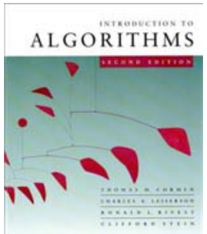

# Example of Dijkstra's algorithm

"*B*" ← **EXTRACT-MIN(*Q*):**



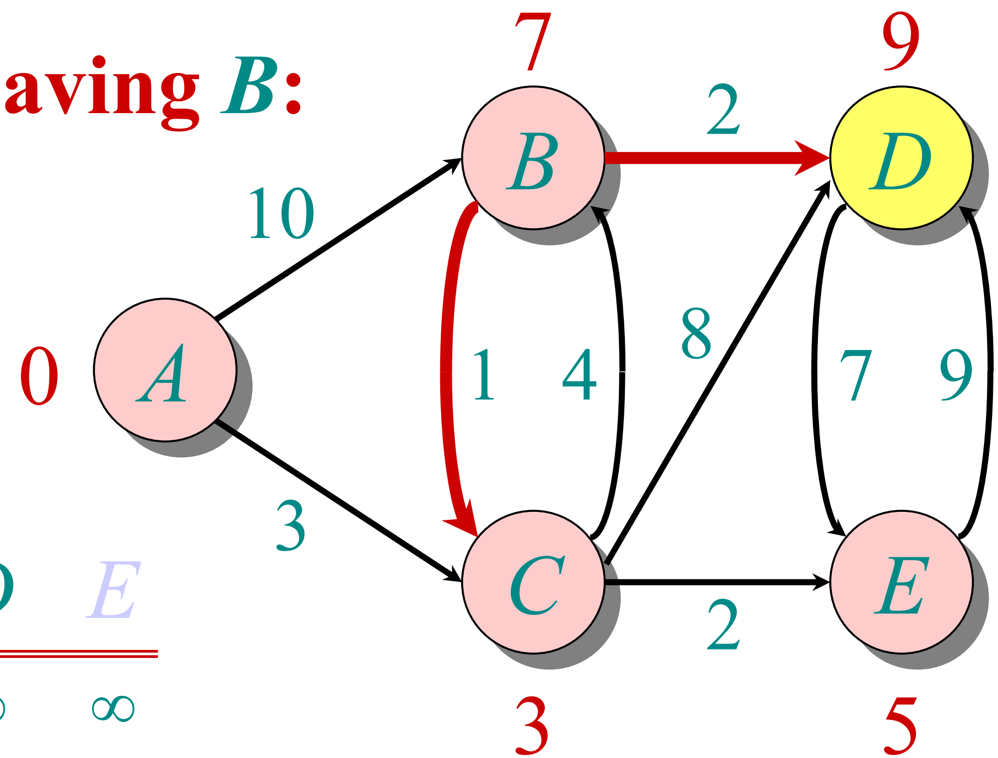

| *Q*: | *A* | *B* | *C* | *D* | *E* |
|---|---|---|---|---|---|
| | 0 | ∞ | ∞ | ∞ | ∞ |
| | | 10 | 3 | ∞ | ∞ |
| | | 7 | | 11 | 5 |
| | | 7 | | 11 | |

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



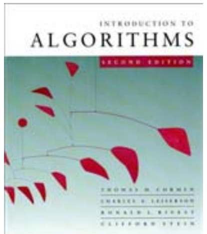

# Example of Dijkstra's algorithm

**Relax all edges leaving *B*:**



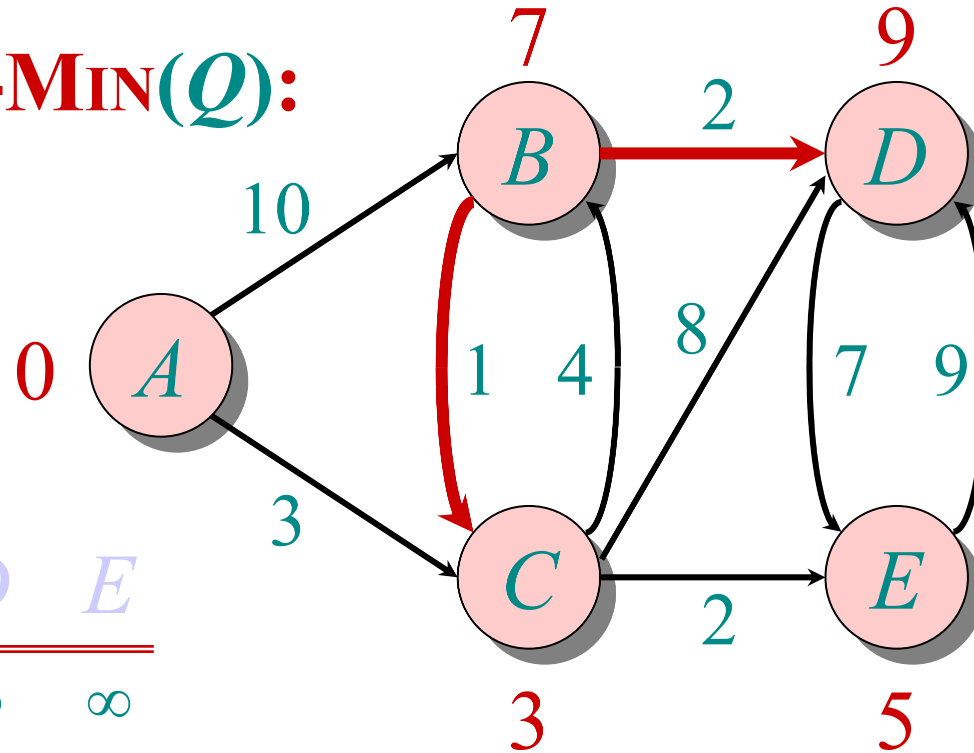

| *Q*: | *A* | *B* | *C* | *D* | *E* |
|---|---|---|---|---|---|
| | 0 | ∞ | ∞ | ∞ | ∞ |
| | | 10 | 3 | ∞ | ∞ |
| | | 7 | | 11 | 5 |
| | | 7 | | 11 | |
| | | | | 9 | |

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



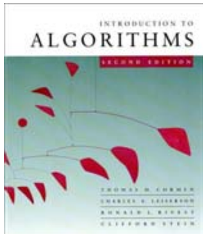

# Example of Dijkstra's algorithm

*"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* }




# Analysis of Dijkstra

$|V|$ times:

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

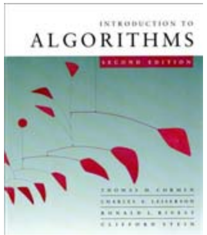
$degree(u)$ times:

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

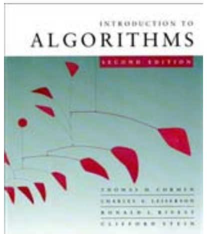
**Note:** Same formula as in the analysis of Prim's minimum spanning tree algorithm.




# Analysis of Dijkstra (continued)

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

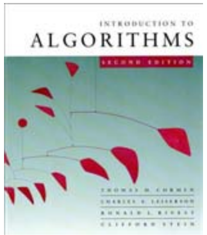
| $Q$ | $T_{\text{EXTRACT-MIN}}$ | $T_{\text{DECREASE-KEY}}$ | Total |
|---|---|---|---|
| array | $O(\|V\|)$ | $O(1)$ | $O(\|V\|^2)$ |
| binary heap | $O(\log \|V\|)$ | $O(\log \|V\|)$ | $O(\|E\| \log \|V\|)$ |
| Fibonacci heap | $O(\log \|V\|)$ amortized | $O(1)$ amortized | $O(\|E\| + \|V\| \log \|V\|)$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




# Correctness

**Theorem.** (i) For all $v \in S$: $d[v] = \delta(s, v)$
(ii) For all $v \notin S$: $d[v]$ = weight of shortest path from $s$ to $v$ that uses only (besides $v$ itself) vertices in $S$.

**Corollary.** Dijkstra's algorithm terminates with $d[v] = \mathrm{d}(s, v)$ for all $v \in V$.

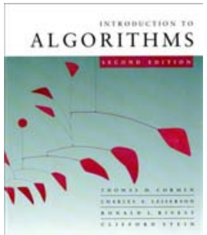



# Correctness

**Theorem.** (i) For all $v \in S$: $d[v] = \delta(s, v)$
(ii) For all $v \notin S$: $d[v]$ = weight of shortest path from $s$ to $v$ that uses only (besides $v$ itself) vertices in $S$.

*Proof.* By induction.

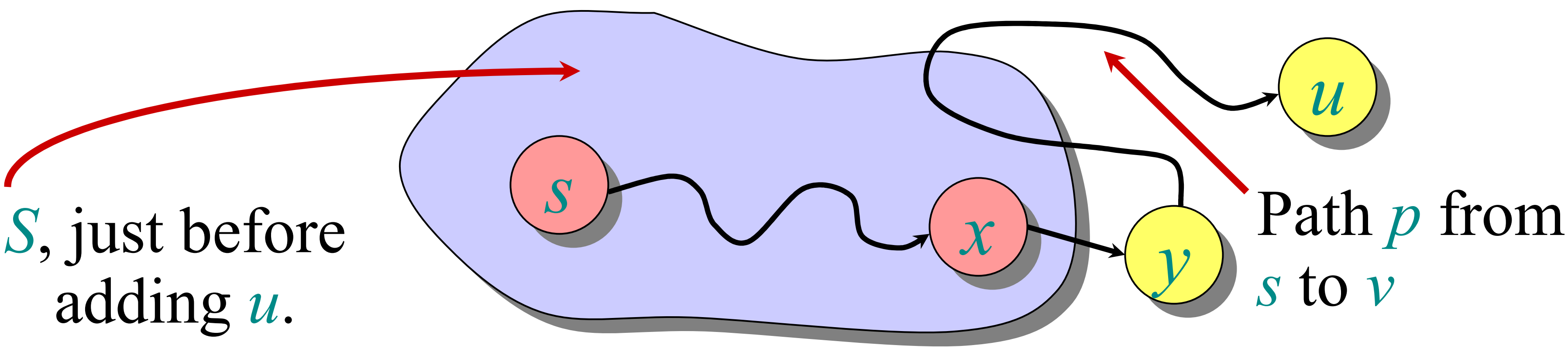
- Base: Before the while loop, $d[s]=0$ and $d[v]=\infty$ for all $v \neq s$, so (i) and (ii) are true.
- Step: Assume (i) and (ii) are true before an iteration; now we need to show they are remain true after another iteration. Let $u$ be the vertex added to $S$, so $d[u] \leq d[v]$ for all other $v \notin S$.

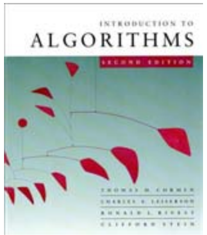



# Correctness

**Theorem.** (i) For all $v \in S$: $d[v] = \delta(s, v)$
(ii) For all $v \notin S$: $d[v]$ = weight of shortest path from $s$ to $v$ that uses only (besides $v$ itself) vertices in $S$.

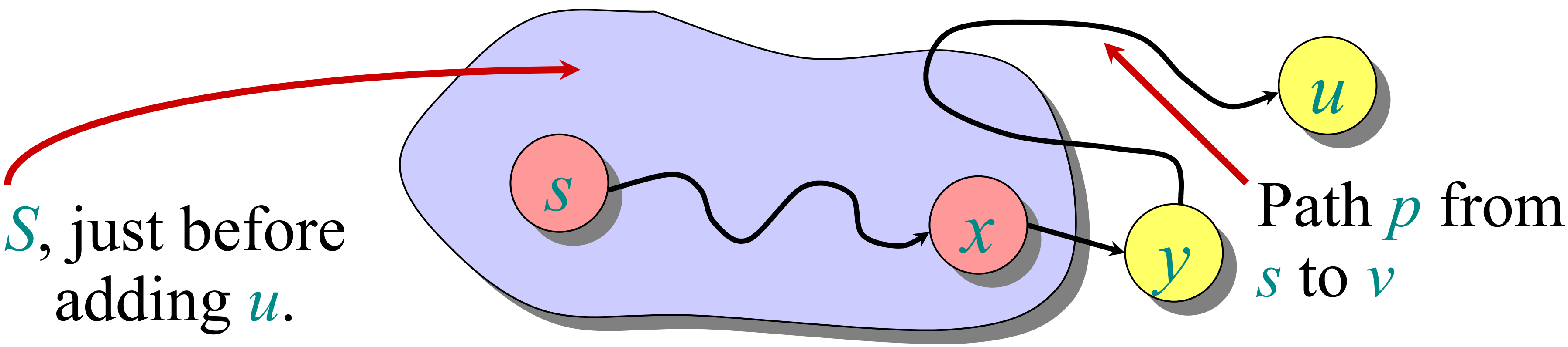
- (i) Need to show that $d[u] = \delta(s, u)$. Assume the contrary.
$\Rightarrow$ There is a path $p$ from s to u with $w(p) < d[u]$ that uses vertices $\notin S$.
$\Rightarrow$ Let $y$ be first vertex on $p$ such that $y \notin S$.



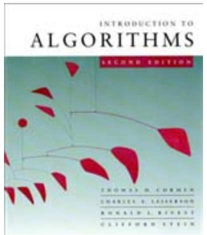



# Correctness

**Theorem.** (i) For all $v \in S$: $d[v] = \delta(s, v)$
(ii) For all $v \notin S$: $d[v]$ = weight of shortest path from $s$ to $v$ that uses only (besides $v$ itself) vertices in $S$.



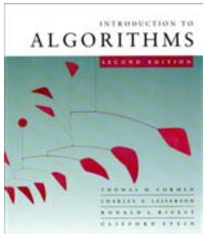

$\Rightarrow d[y] \leq w(p) < d[u]$. Contradiction to the choice of $u$.




# Correctness

**Theorem.** (i) For all $v \in S$: $d[v] = \delta(s, v)$
(ii) For all $v \notin S$: $d[v]$ = weight of shortest path from $s$ to $v$ that uses only (besides $v$ itself) vertices in $S$.

- (ii) Let $v \notin S$. Let $p$ be a shortest path from $s$ to $v$ that uses only (besides $v$ itself) vertices in $S$.
  - $p$ does not contain $u$: (ii) true by inductive hypothesis
  - $p$ contains $u$: $p$ consists of vertices in $S \setminus \{u\}$ and ends with an edge from $u$ to $v$.
    $\Rightarrow w(p)=d[u]+w(u,v)$, which is the value of $d[v]$ after adding $u$. So (ii) is true.



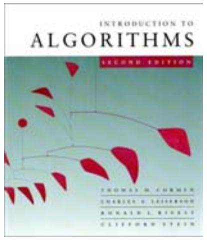

# Unweighted graphs

Suppose $w(u, v) = 1$ for all $(u, v) \in E$. Can the code for Dijkstra be improved?

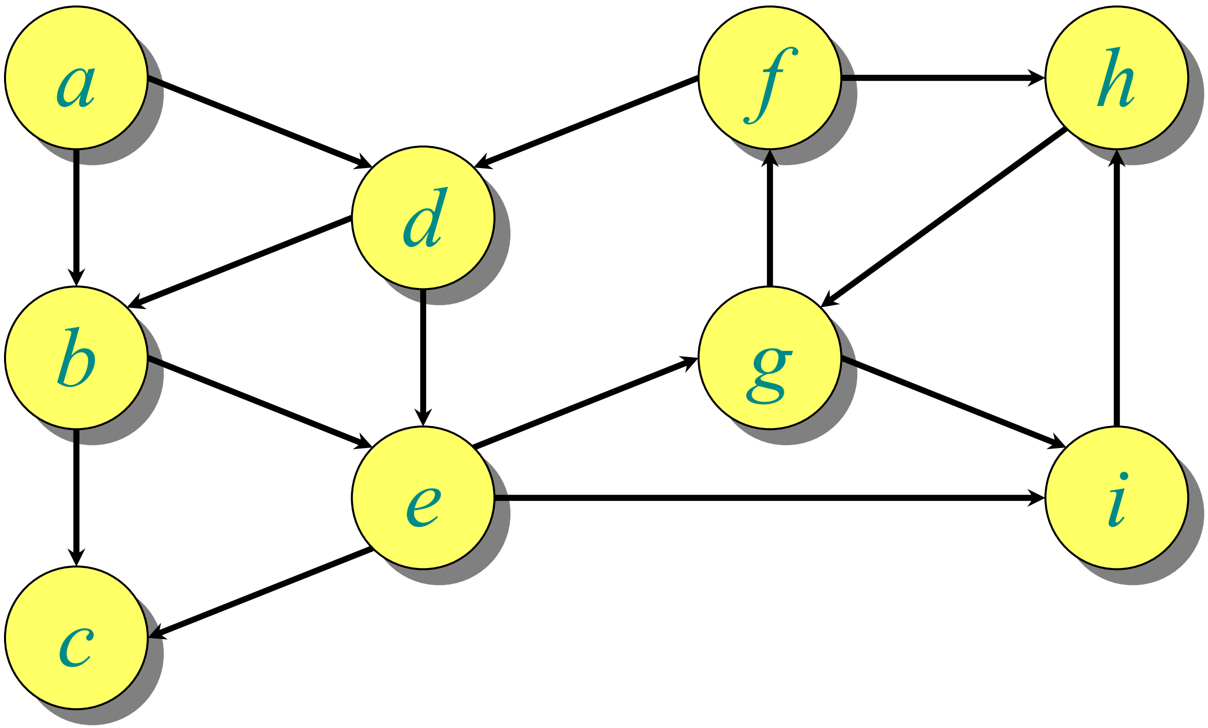
- Use a simple FIFO queue instead of a priority queue.
- ***Breadth-first search***

**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$(Q, v)$

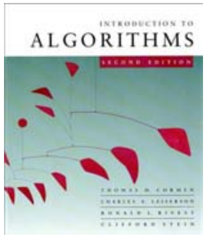
**Analysis:** Time $= O(|V| + |E|)$.



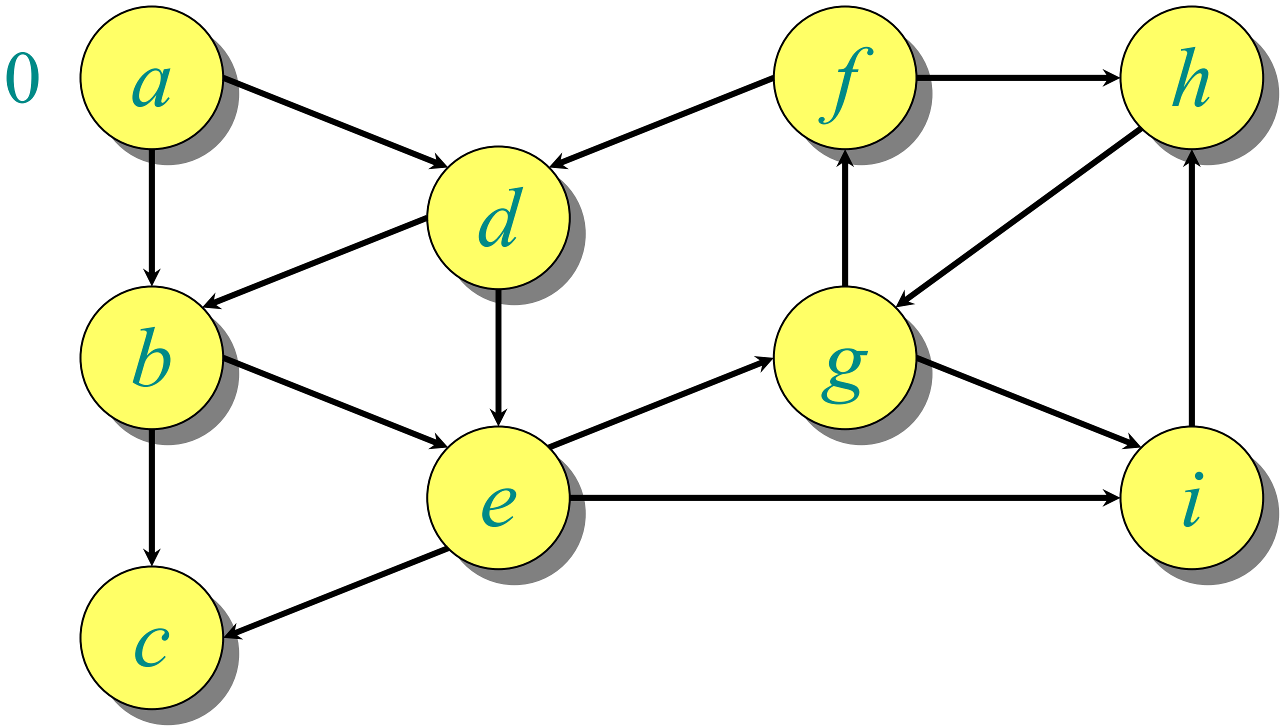
# Example of breadth-first search




*Q*:

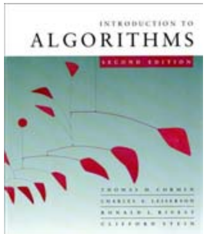


# Example of breadth-first search

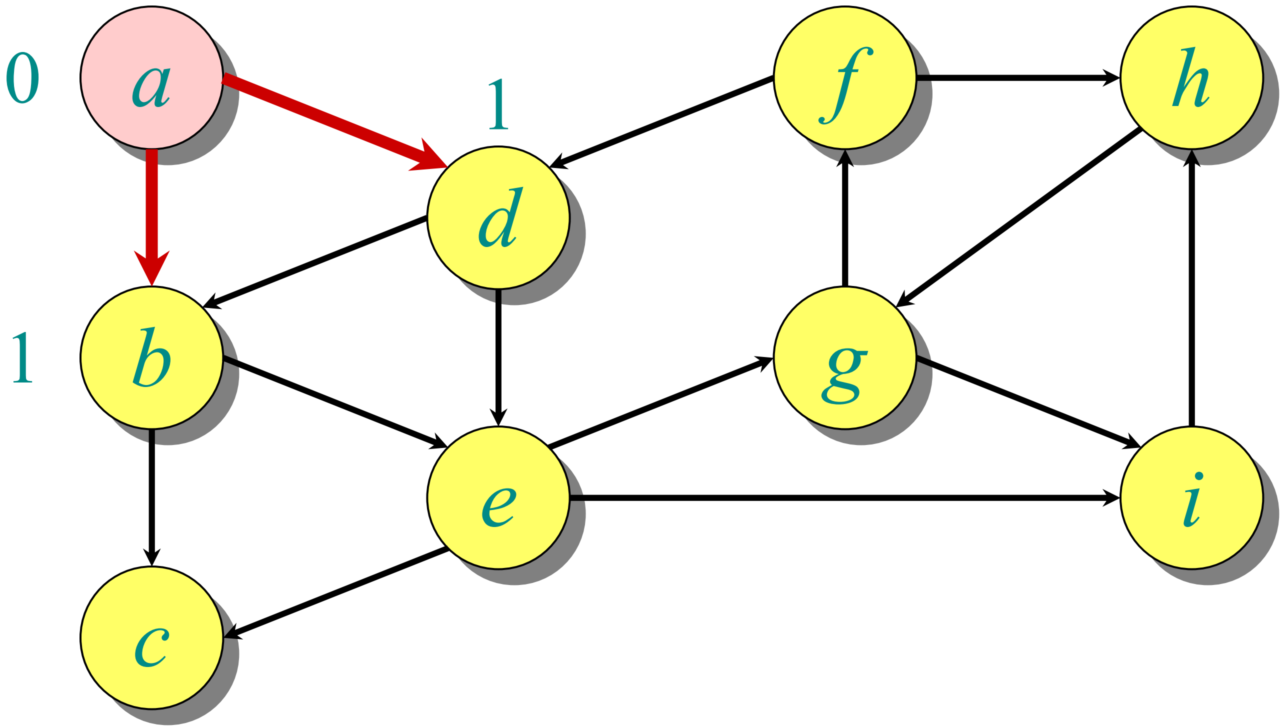



0

*Q*: *a*

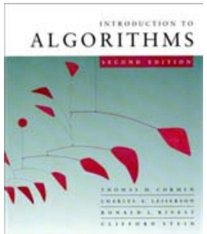



# Example of breadth-first search

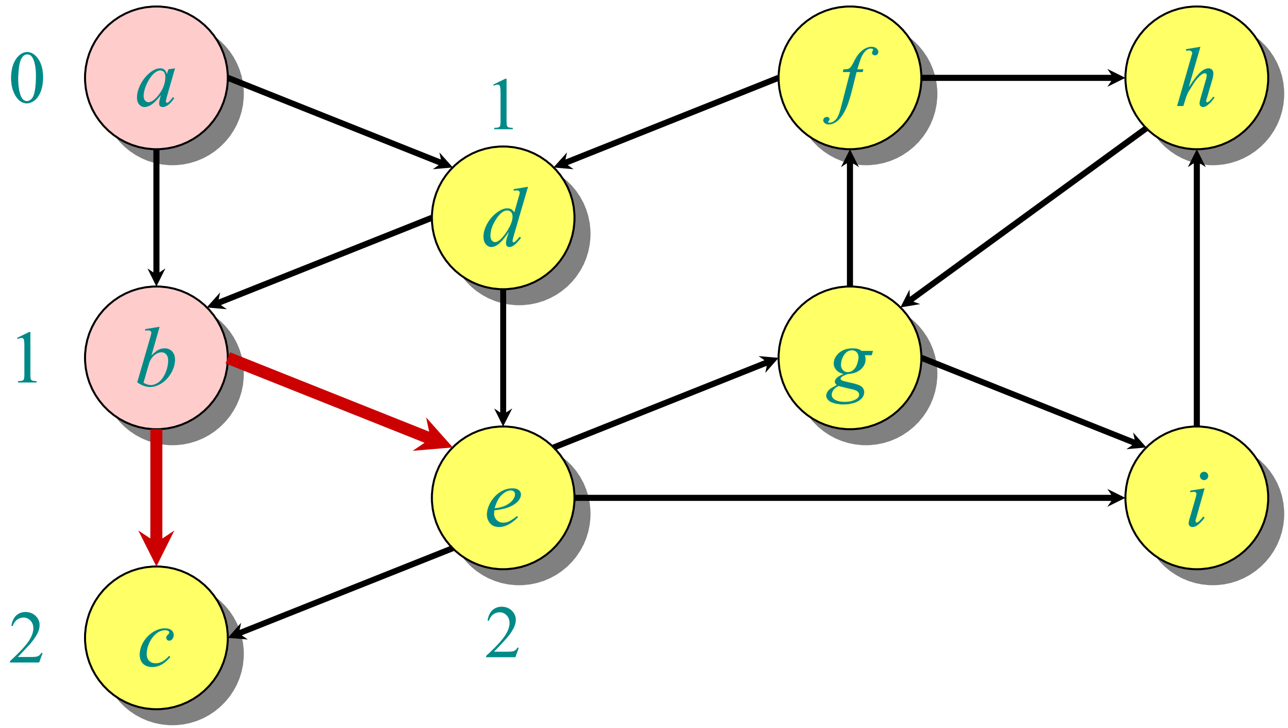



1 1

*Q*: *a* *b* *d*

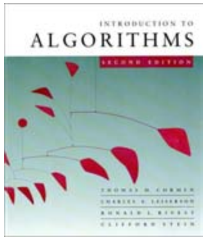



# Example of breadth-first search

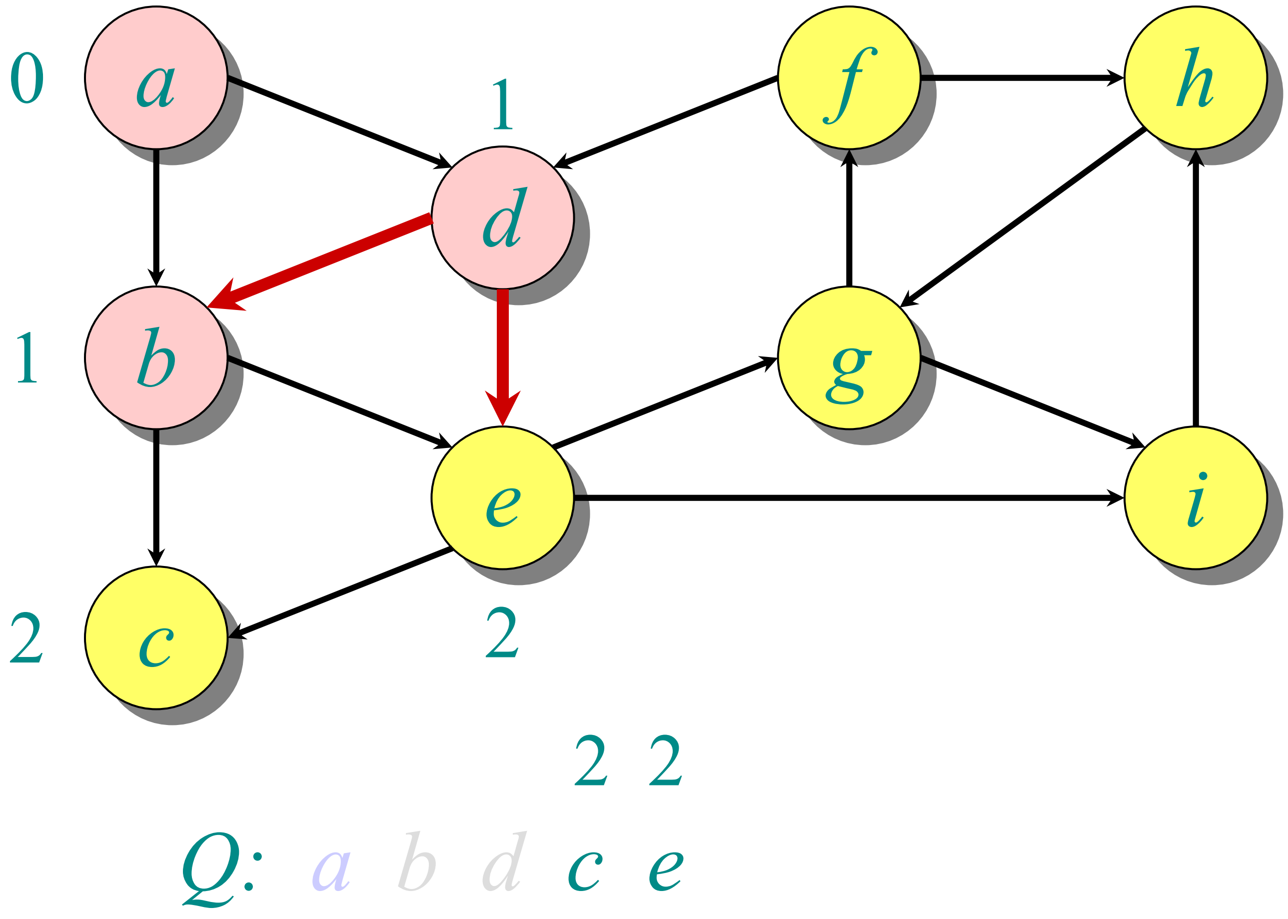


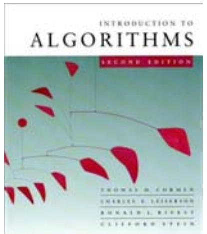

1 2 2

*Q*: *a* *b* *d* *c* *e*

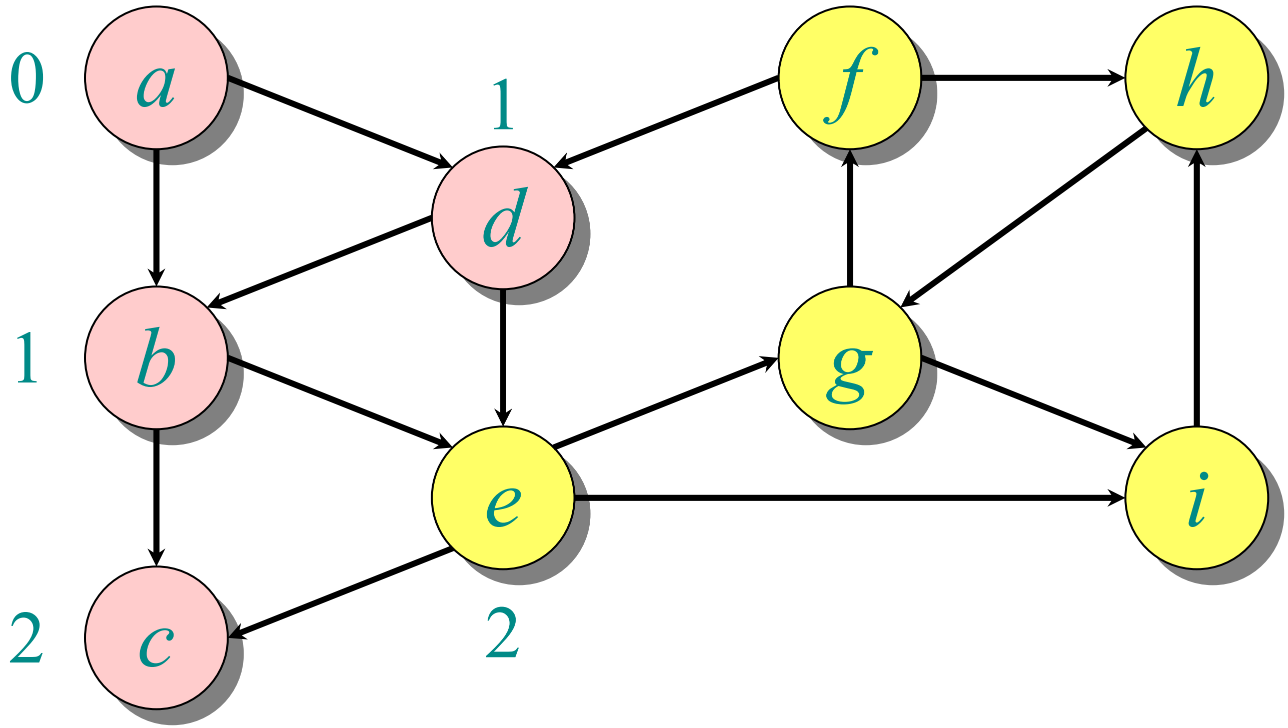



# Example of breadth-first search



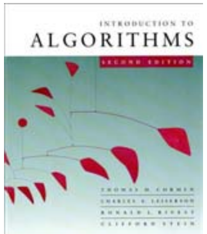



# Example of breadth-first search

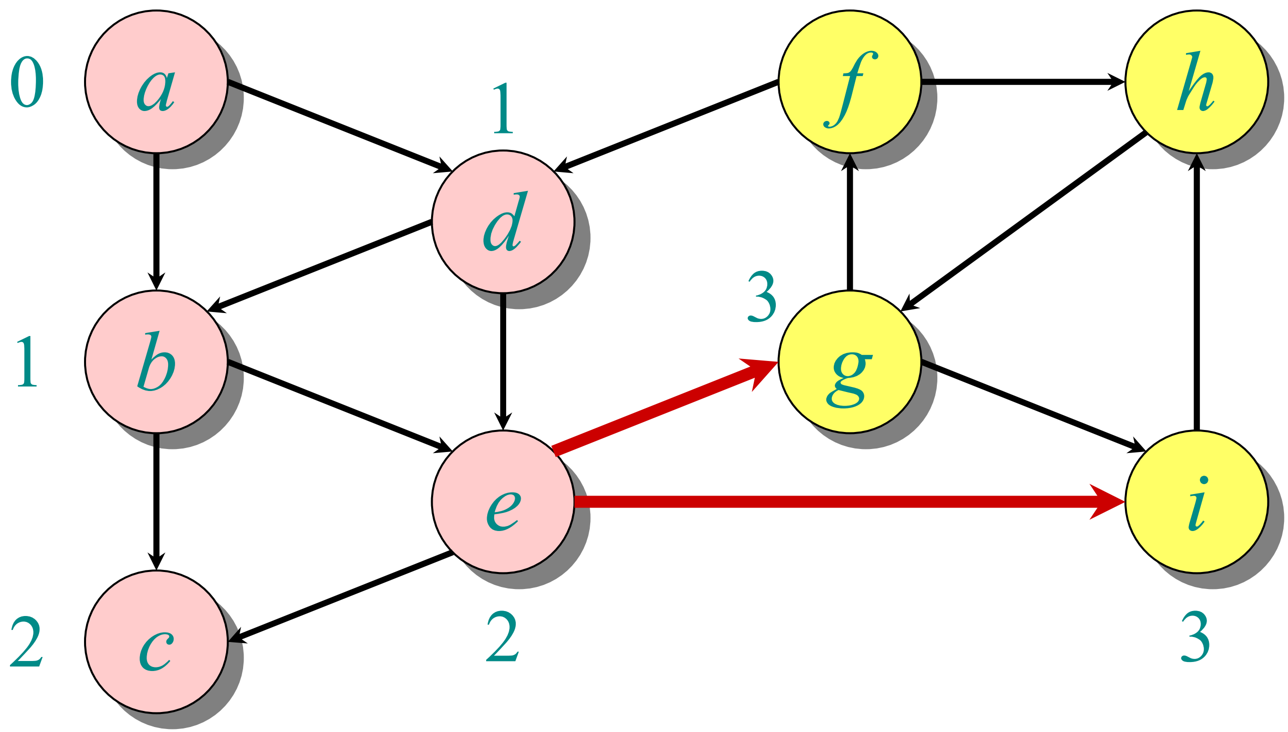



2

*Q*: *a* *b* *d* *c* *e*

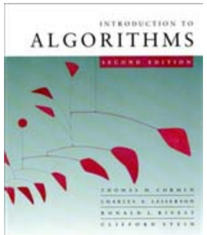



# Example of breadth-first search

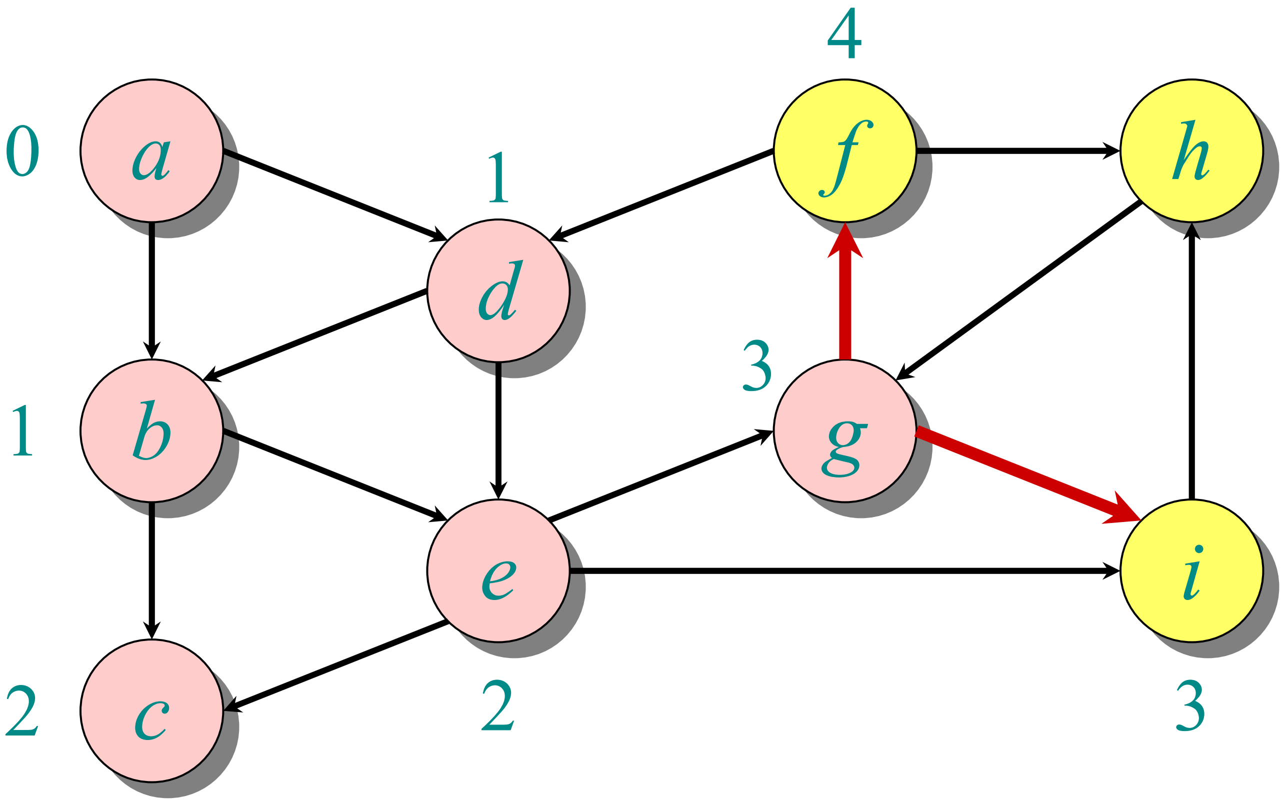



3 3

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



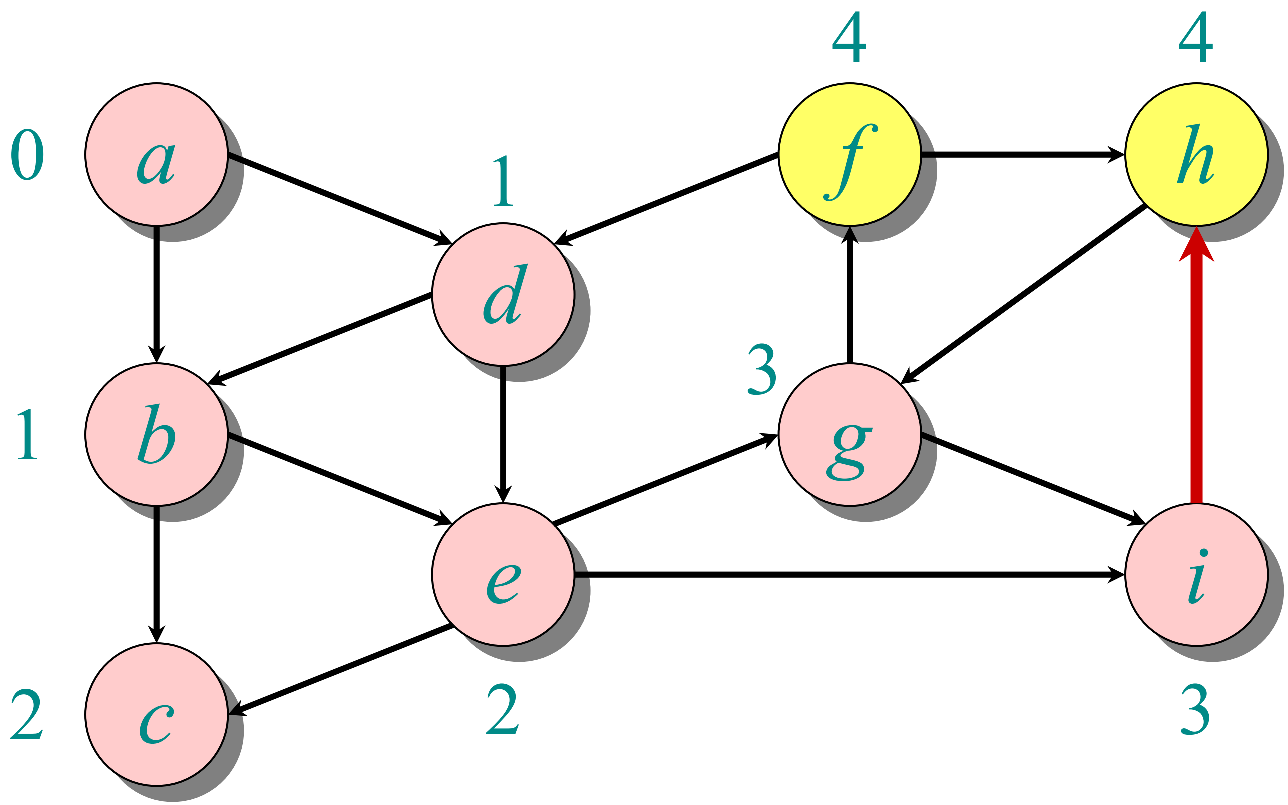

# Example of breadth-first search




3 4

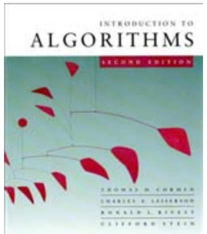
*Q*: *a* *b* *d* *c* *e* *g* *i* *f*

# Example of breadth-first search

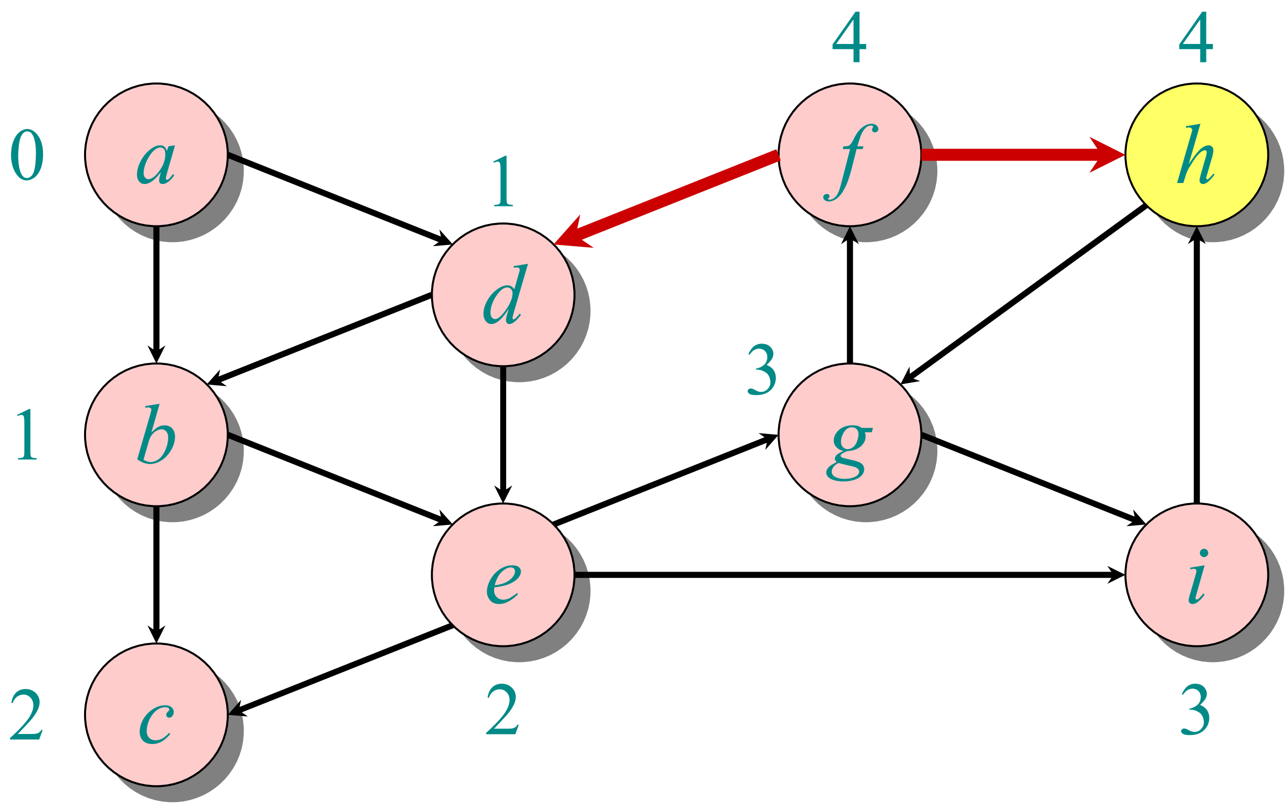



4 4

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

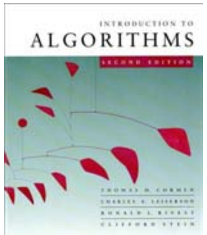



# Example of breadth-first search

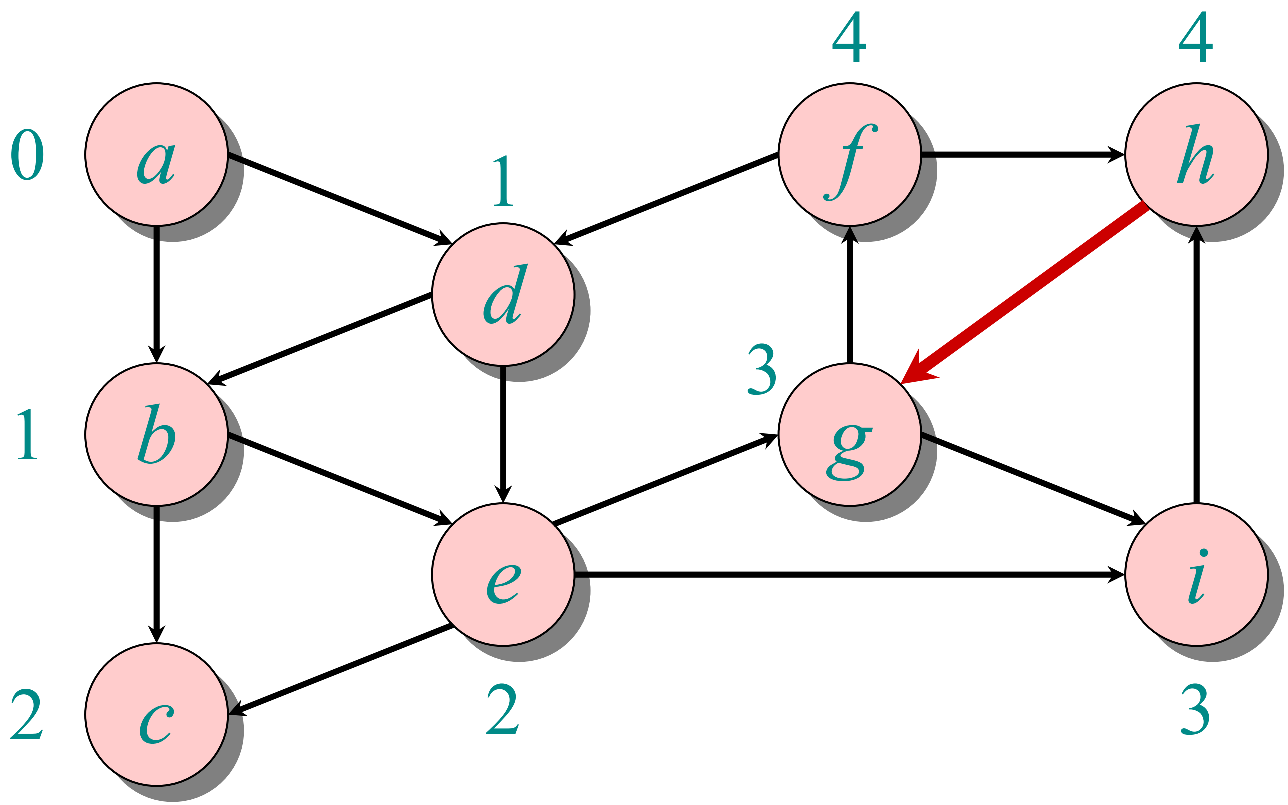



4

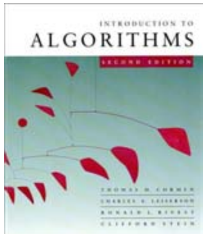
*Q*: *a b d c e g i f h*



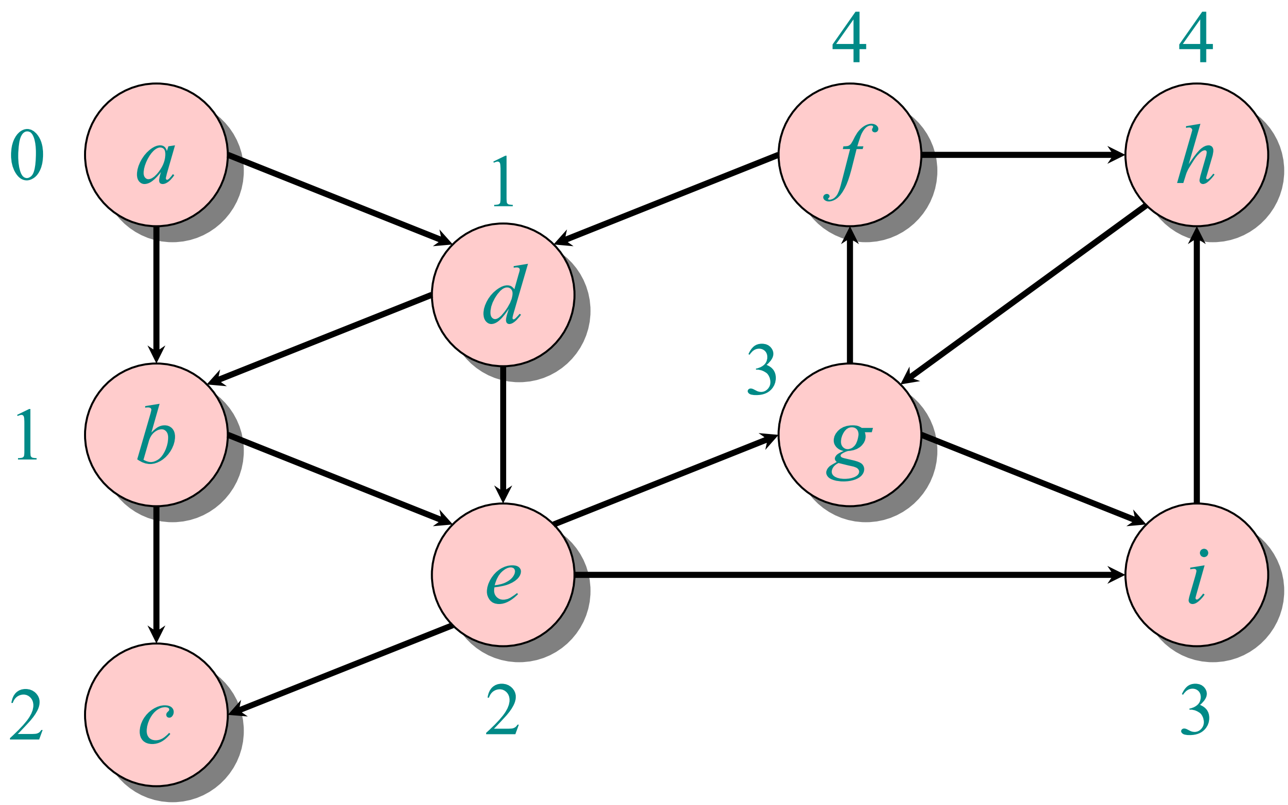

# Example of breadth-first search



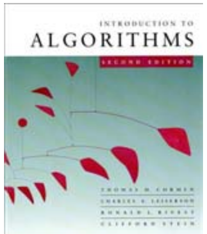

*Q*: *a b d c e g i f h*



# Example of breadth-first search




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



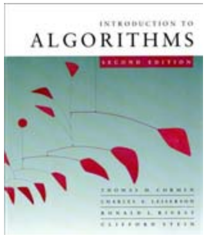

# Correctness of BFS

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

**Key idea:**

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

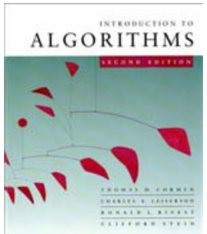
- **Invariant:** $v$ comes after $u$ in $Q$ implies that $d[v] = d[u]$ or $d[v] = d[u] + 1$.




# How to find the actual shortest paths?

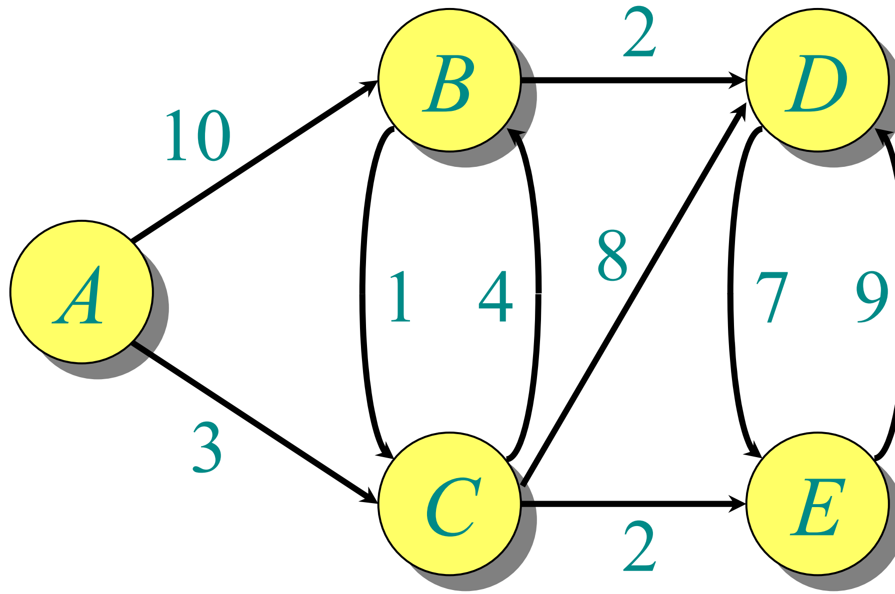
## Store a predecessor tree:

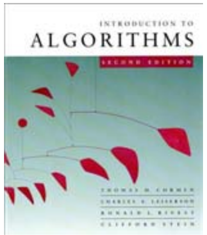
$d[s] \leftarrow 0$
**for** each $v \in V - \{s\}$
  **do** $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$   $\triangleright$ $Q$ is a priority queue maintaining $V - S$
**while** $Q \neq \varnothing$
  **do** $u \leftarrow$ EXTRACT-MIN$(Q)$
    $S \leftarrow S \cup \{u\}$
    **for**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



# Example of Dijkstra's algorithm

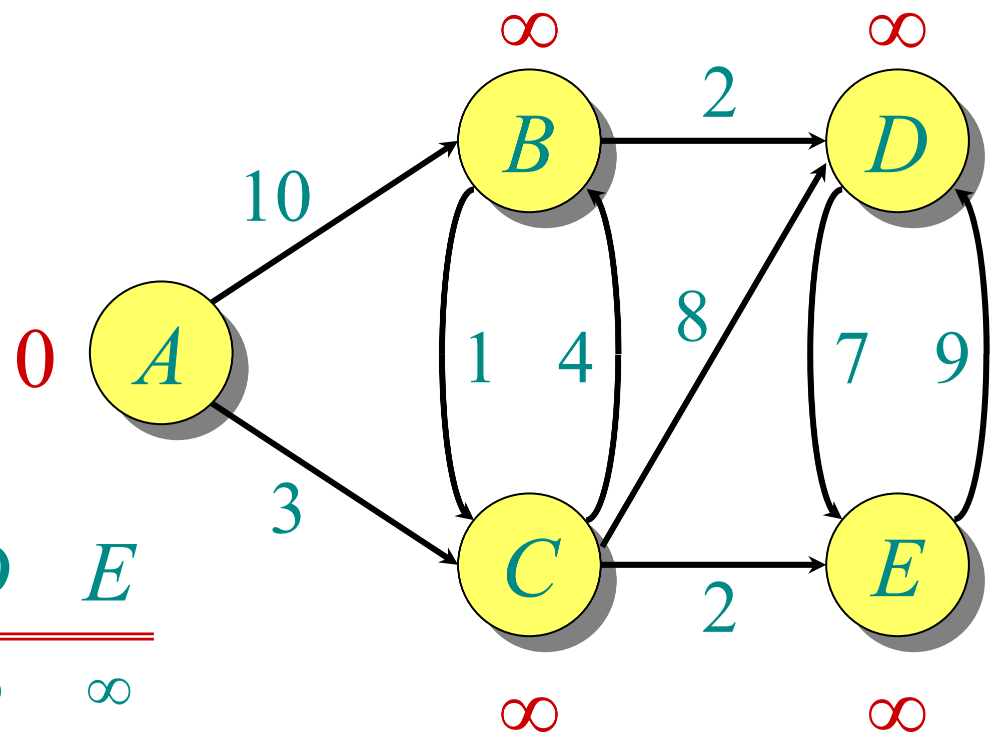
**Graph with nonnegative edge weights:**






# Example of Dijkstra's algorithm

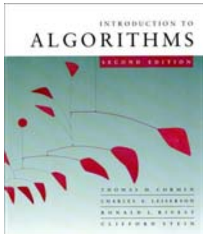
**Initialize:**




*Q*: *A* *B* *C* *D* *E*

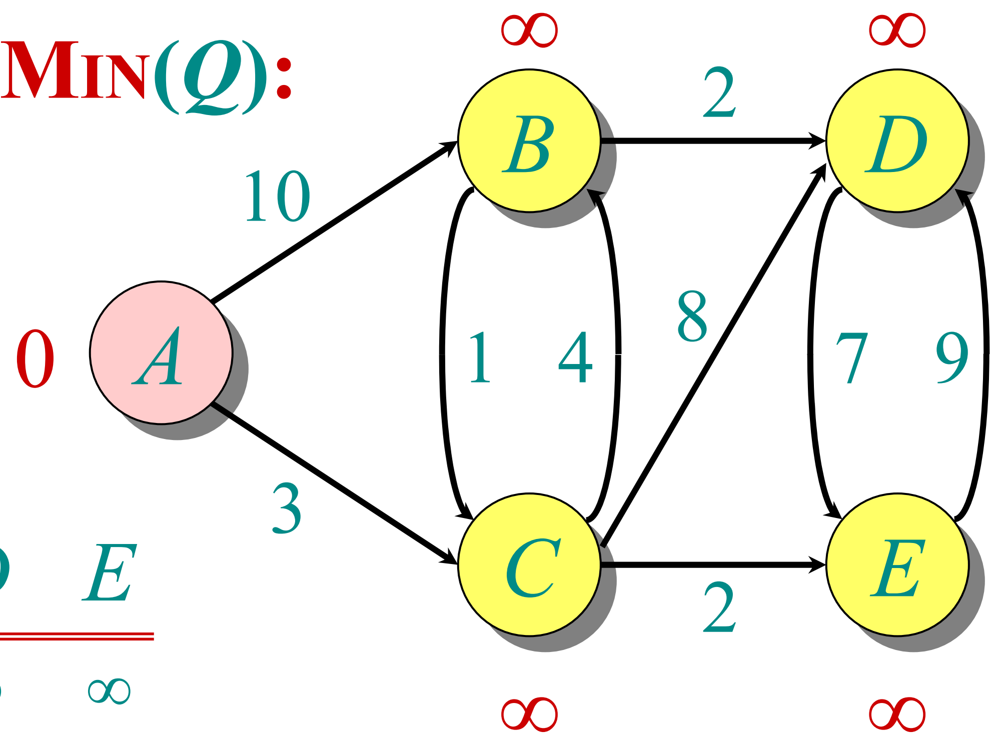
| *A* | *B* | *C* | *D* | *E* |
|---|---|---|---|---|
| 0 | ∞ | ∞ | ∞ | ∞ |

*S*: {}



# Example of Dijkstra's algorithm

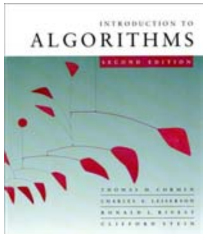
"$A$" ← EXTRACT-MIN($Q$):




| $Q$: | $A$ | $B$ | $C$ | $D$ | $E$ |
|---|---|---|---|---|---|
| | 0 | ∞ | ∞ | ∞ | ∞ |

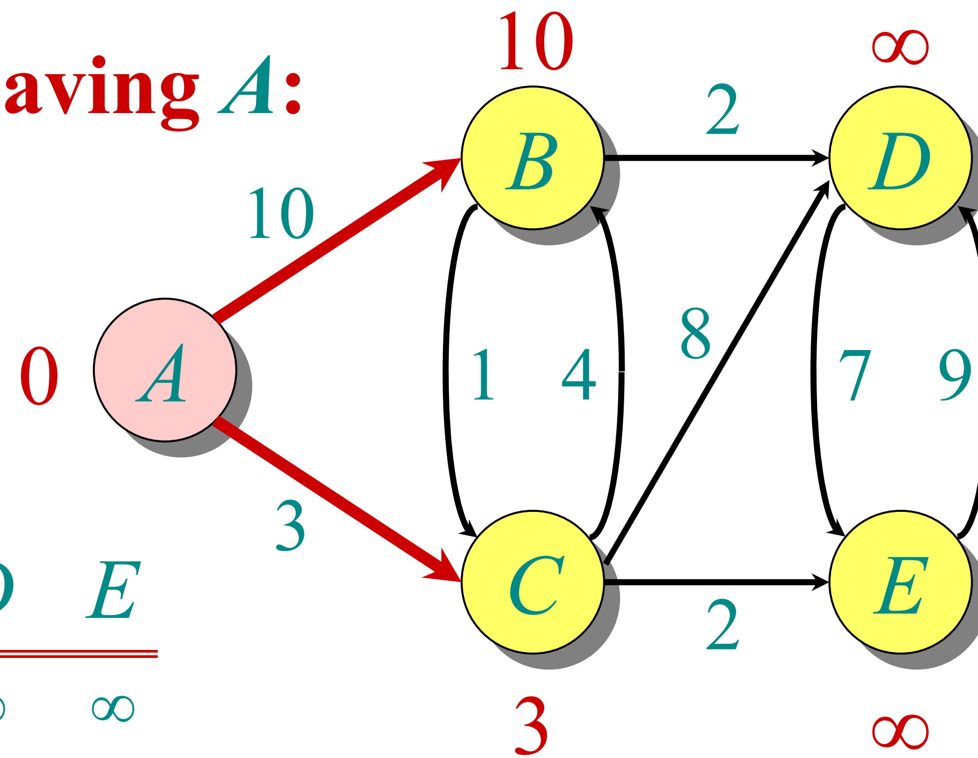
$S$: { $A$ }

| π: | $A$ | $B$ | $C$ | $D$ | $E$ |
|---|---|---|---|---|---|
| | – | – | – | – | – |




# Example of Dijkstra's algorithm

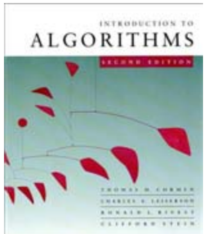
**Relax all edges leaving *A*:**




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

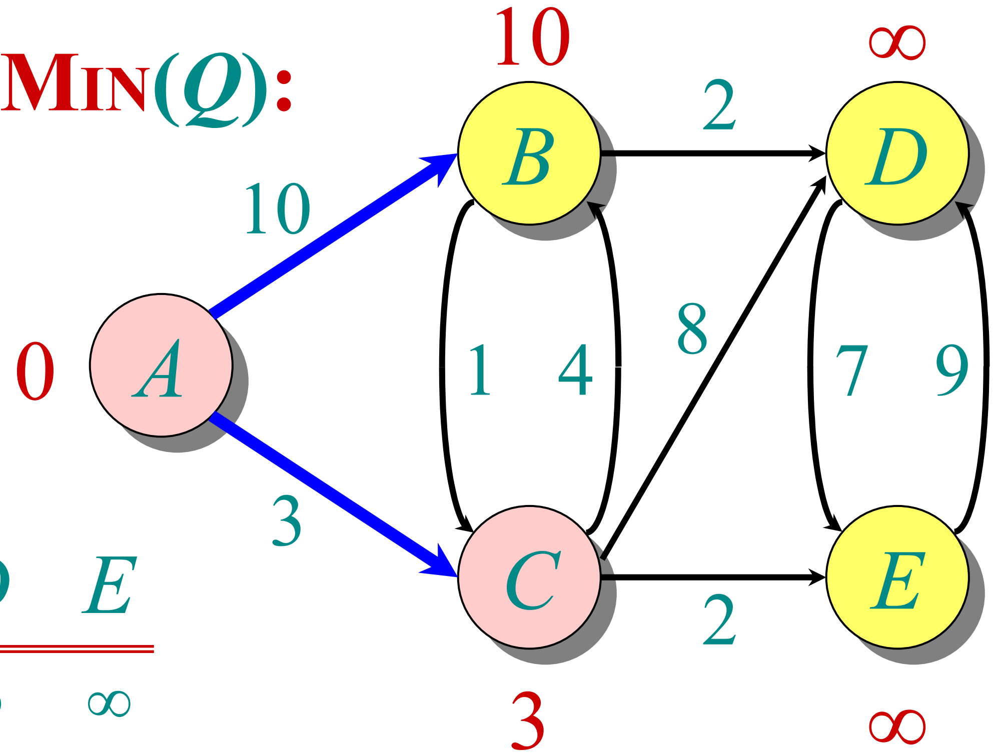
*S*: { *A* }

| π: | *A* | *B* | *C* | *D* | *E* |
|---|---|---|---|---|---|
| | – | A | A | – | - |




# Example of Dijkstra's algorithm

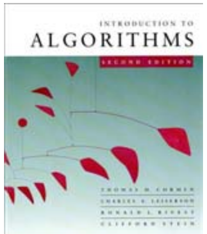
"*C*" ← EXTRACT-MIN(*Q*):




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

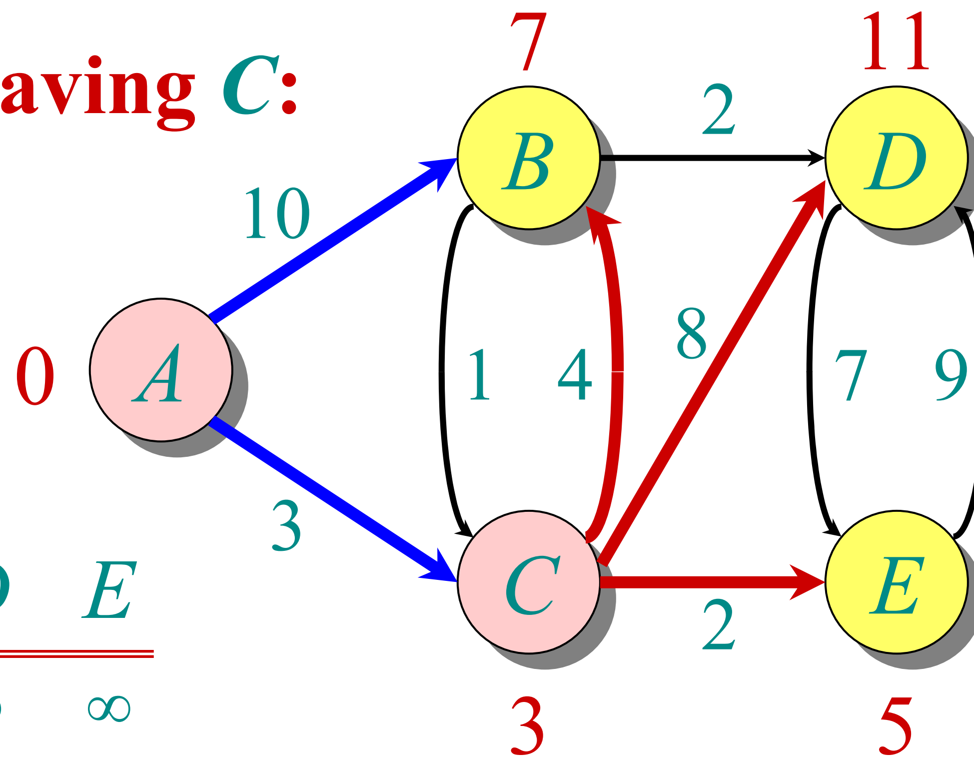
*S*: { *A*, *C* }

| π: | *A* | *B* | *C* | *D* | *E* |
|---|---|---|---|---|---|
| | – | A | A | – | - |




# Example of Dijkstra's algorithm

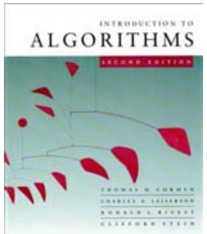
**Relax all edges leaving *C*:**




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

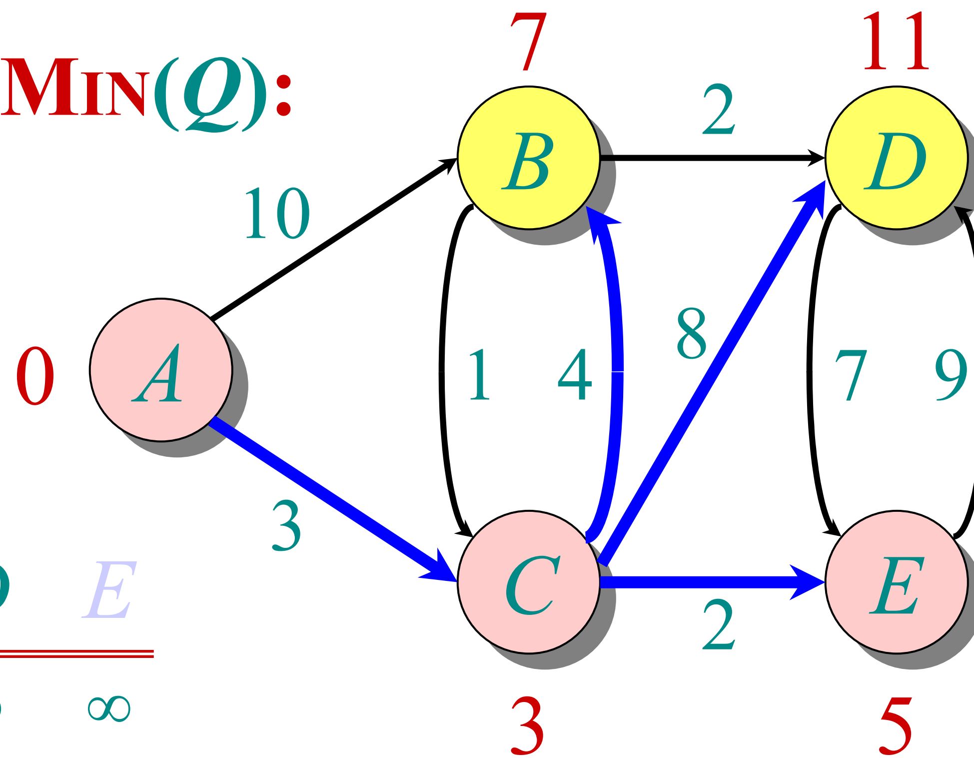
*S*: { *A*, *C* }

| π: | *A* | *B* | *C* | *D* | *E* |
|---|---|---|---|---|---|
| | – | C | A | C | C |




# Example of Dijkstra's algorithm

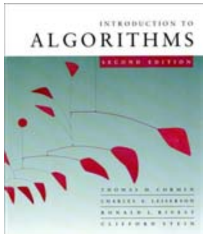
"*E*" ← EXTRACT-MIN(*Q*):




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

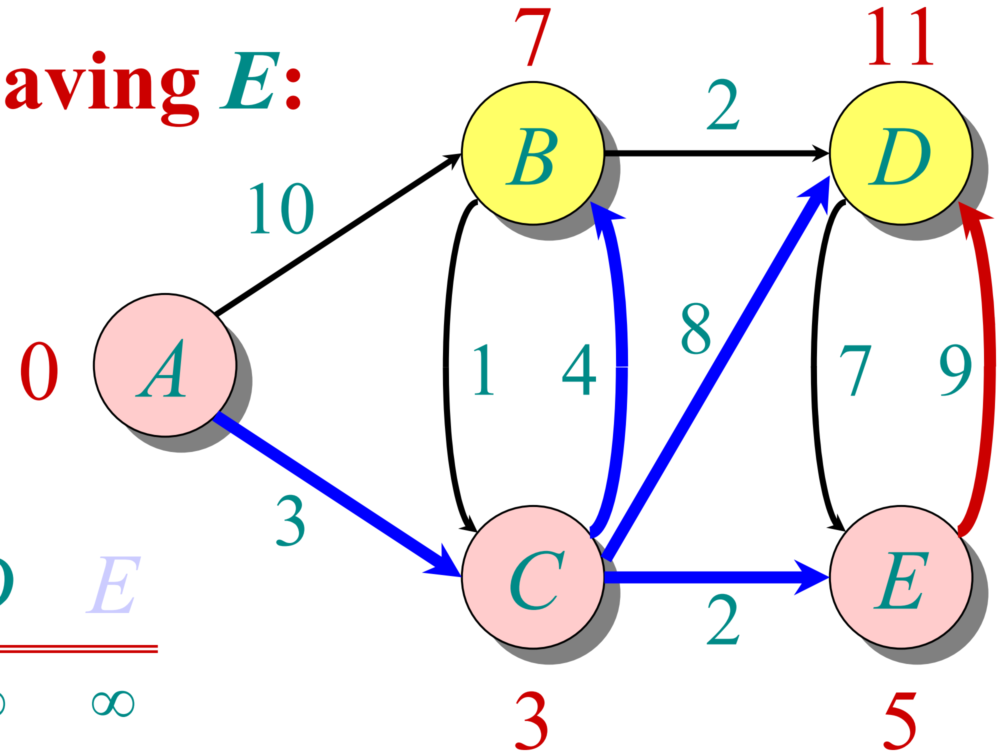
*S*: { *A*, *C*, *E* }

| π: | *A* | *B* | *C* | *D* | *E* |
|---|---|---|---|---|---|
| | – | C | A | C | C |




# Example of Dijkstra's algorithm

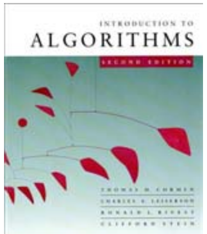
**Relax all edges leaving *E*:**




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

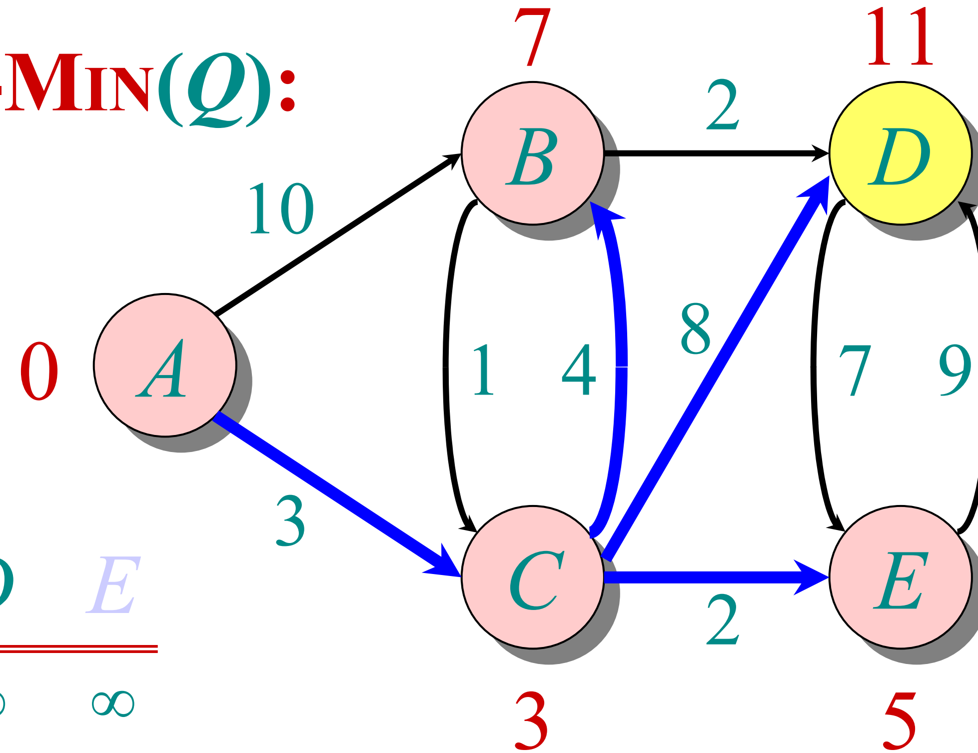
*S*: { *A*, *C*, *E* }

| π: | *A* | *B* | *C* | *D* | *E* |
|---|---|---|---|---|---|
| | — | C | A | C | C |



# Example of Dijkstra's algorithm

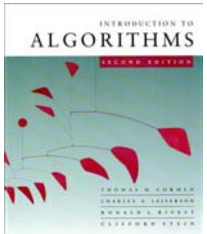
"*B*" ← EXTRACT-MIN(*Q*):




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

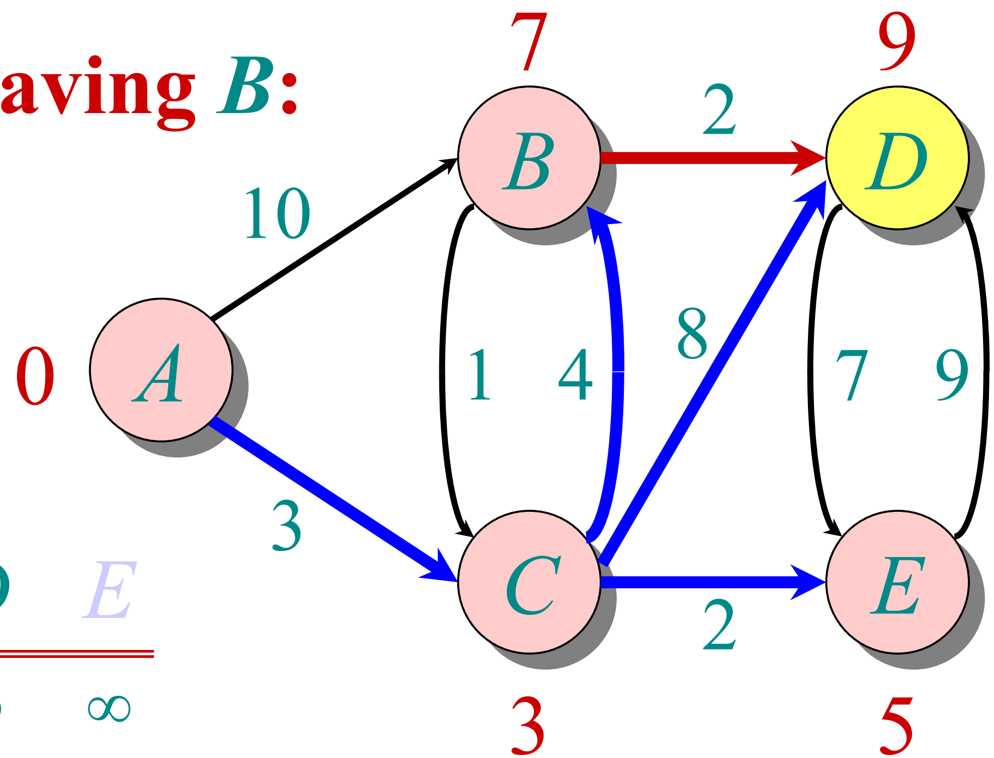
*S*: { *A*, *C*, *E*, *B* }

| π: | *A* | *B* | *C* | *D* | *E* |
|---|---|---|---|---|---|
| | – | C | A | C | C |




# Example of Dijkstra's algorithm

**Relax all edges leaving *B*:**



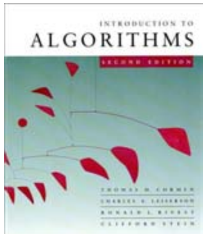

*Q*:

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

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

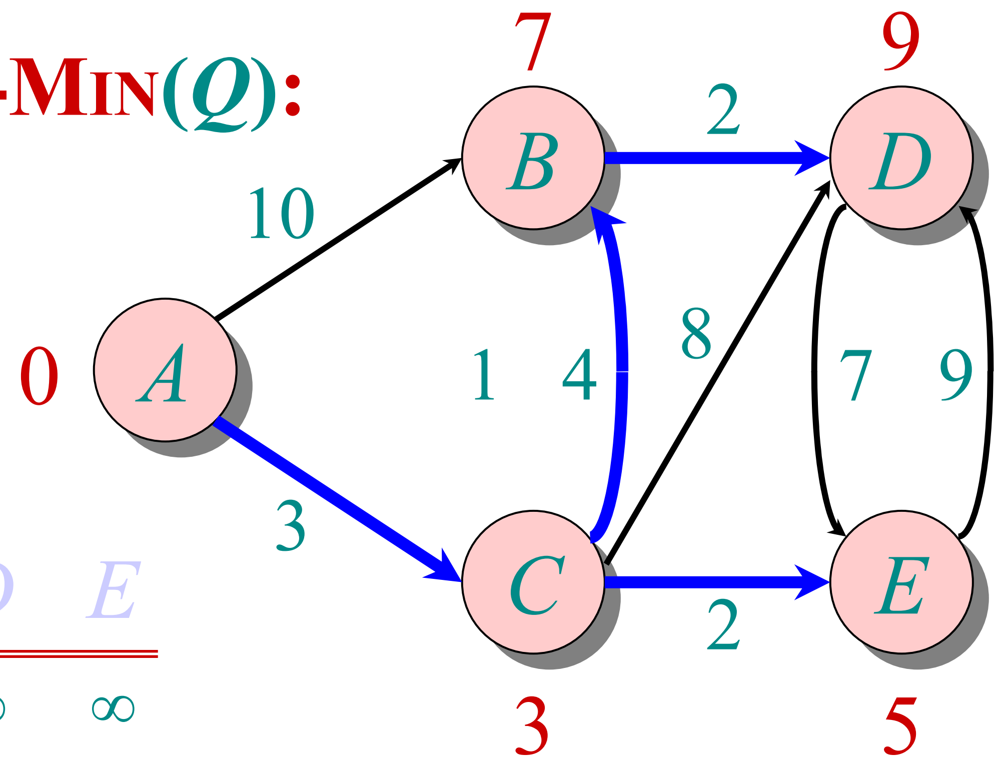
π:

| *A* | *B* | *C* | *D* | *E* |
|---|---|---|---|---|
| | C | A | B | C |




# 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* }

| π: | *A* | *B* | *C* | *D* | *E* |
|---|---|---|---|---|---|
| | — | C | A | B | C |