Single Source Shortest Paths

Carola Wenk

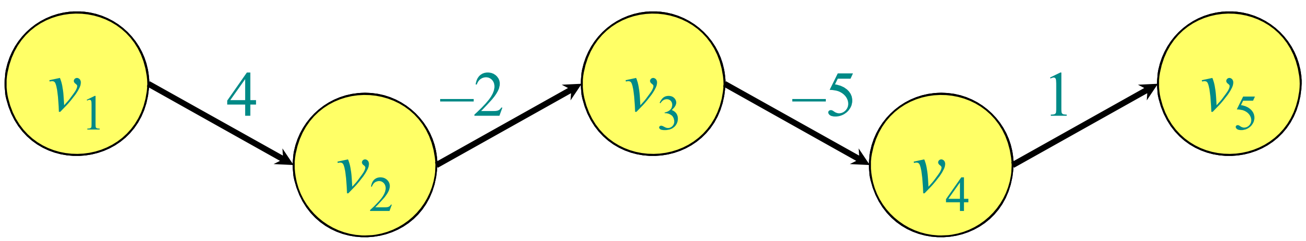
Slides courtesy of Charles Leiserson with changes and additions by Carola Wenk

Paths in graphs

Consider a digraph $G = (V, E)$ with an 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

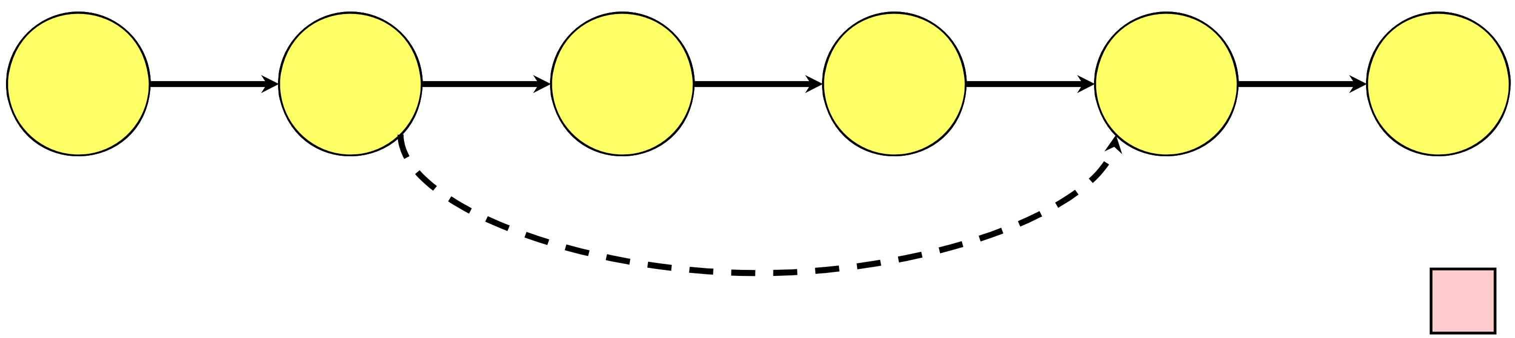
$\delta(u, v) = \min\{w(p) : p$ is a path from u to $v\}$.

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:



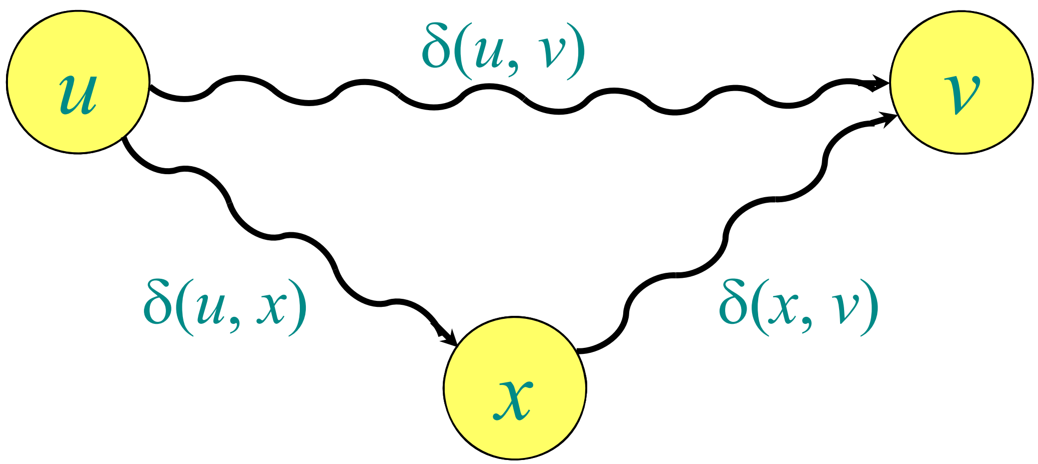
Triangle inequality

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

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

Proof.

- $\delta(u,v)$ minimizes over **all** paths from u to v
- Concatenating two shortest paths from u to x and from x to v yields **one** specific path from u to v

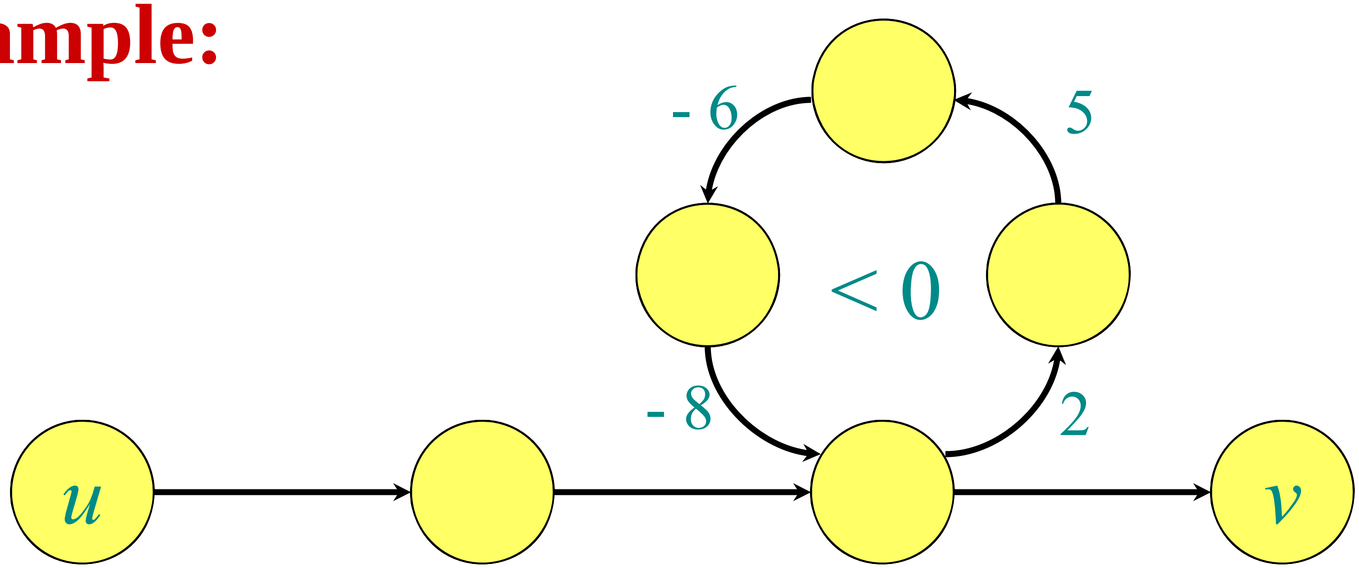


□

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 $v \in V$.

Assumption:
All edge weights $w(u, v)$ are ***non-negative***.

It follows that all shortest-path weights must exist.

IDEA: Greedy.

1. Maintain a set S of vertices whose shortest-path weights from s are known, i.e., $d[v]=\delta(s,v)$
2. At each step add to S the vertex $u \in V - S$ whose distance estimate $d[u]$ from s is minimal.
3. Update the distance estimates $d[v]$ of vertices v adjacent to u.

Dijkstra's algorithm

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
  do $d[v] \leftarrow \infty$
$S \leftarrow \emptyset$ ⊳ Vertices for which $d[v]=d(s,v)$
$Q \leftarrow V$ ⊳ Q is a priority queue maintaining $V - S$ sorted by d-values $d[v]$
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**
      $d[v] \leftarrow d[u] + w(u, v)$

relaxation step

implicit DECREASE-KEY in Q

Dijkstra

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
  do $d[v] \leftarrow \infty$
$S \leftarrow \emptyset$ ⊳ Verti
$Q \leftarrow V$ ⊳ Q is sorte
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**
      $d[v] \leftarrow d[u] + w(u, v)$

implicit DECREASE-KEY in Q

PRIM's algorithm

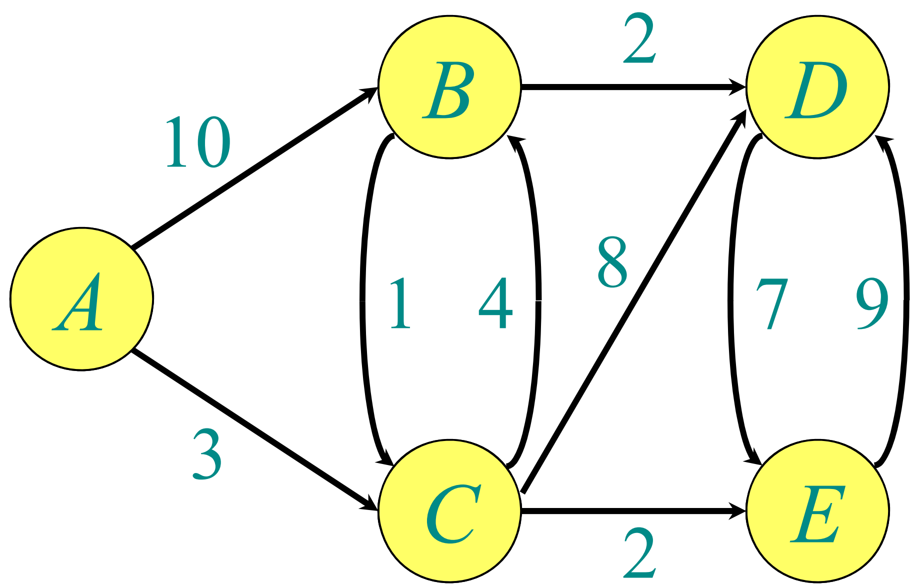
$Q \leftarrow V$
$key[v] \leftarrow \infty$ for all $v \in V$
$key[s] \leftarrow 0$ for some arbitrary $s \in V$
while $Q \neq \emptyset$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    for each $v \in Adj[u]$
      do if $v \in Q$ and $w(u, v) < key[v]$
        then $key[v] \leftarrow w(u, v)$
          $\pi[v]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

Difference to Prim's:
- It suffices to only check $v \in Q$, but it doesn't hurt to check all v
- Add $d[u]$ to the weight

relaxation step

Example of Dijkstra's algorithm

Graph with nonnegative edge weights:

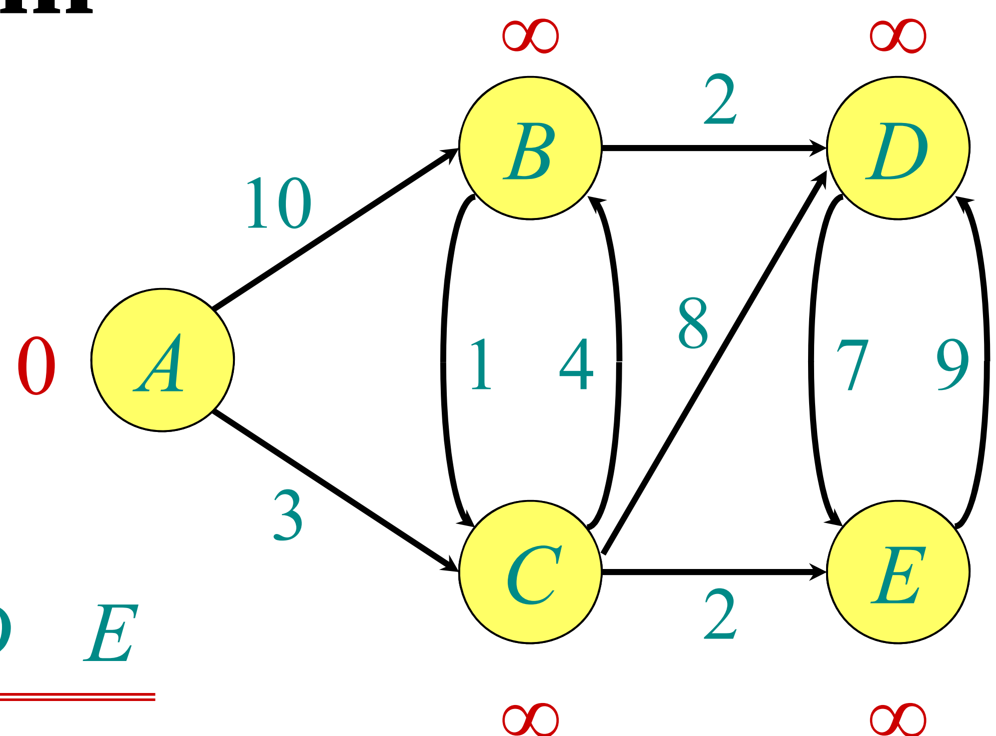


while $Q \neq \emptyset$ **do**
  $u \leftarrow$ EXTRACT-MIN(Q)
  $S \leftarrow S \cup \{u\}$
  for each $v \in Adj[u]$ **do**
    if $d[v] > d[u] + w(u, v)$ **then**
      $d[v] \leftarrow d[u] + w(u, v)$

Example of Dijkstra's algorithm

Initialize:

S: {}

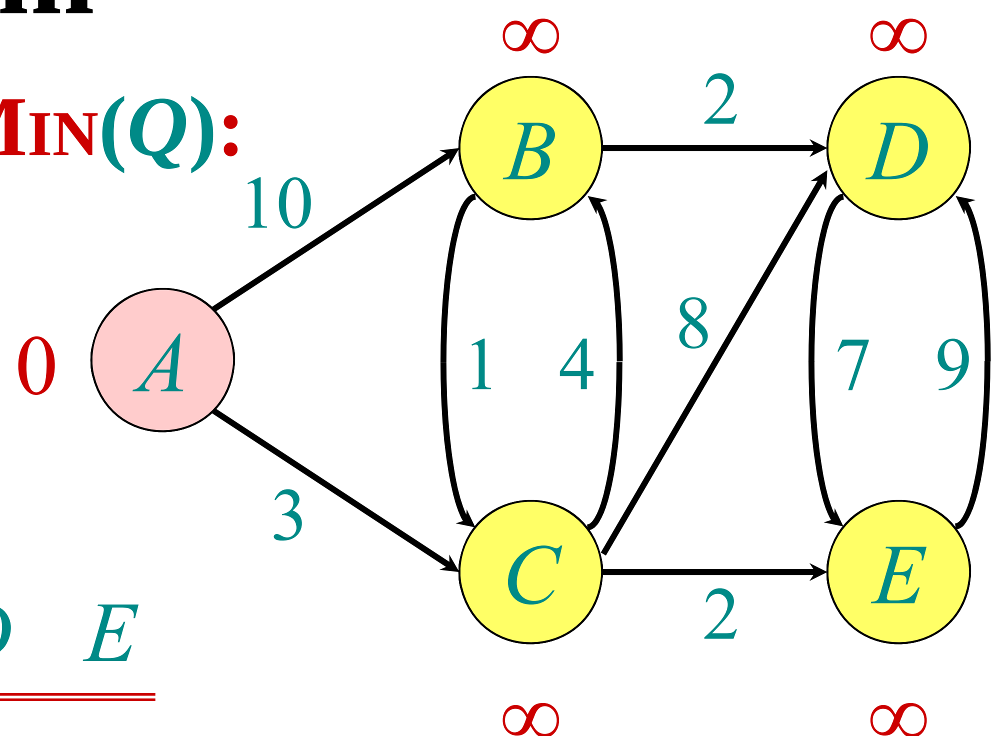


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

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

Example of Dijkstra's algorithm

"A" ← EXTRACT-MIN(Q):



S: { A }

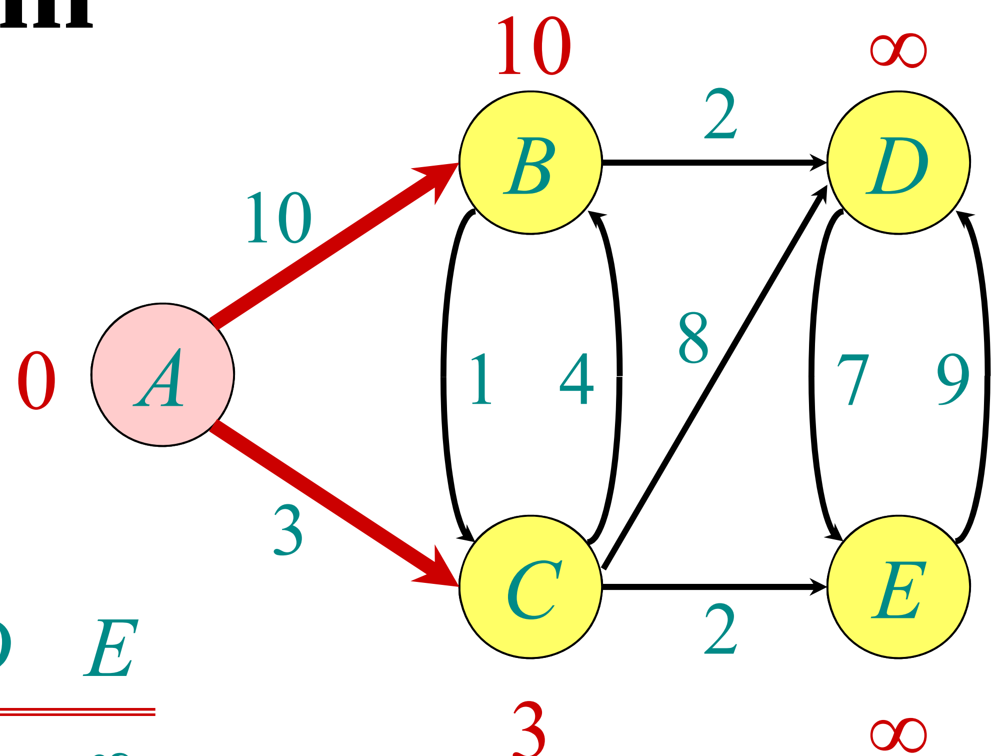
Q:	A	B	C	D	E
	0	∞	∞	∞	∞

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

Example of Dijkstra's algorithm

Relax all edges leaving *A*:

S: { *A* }



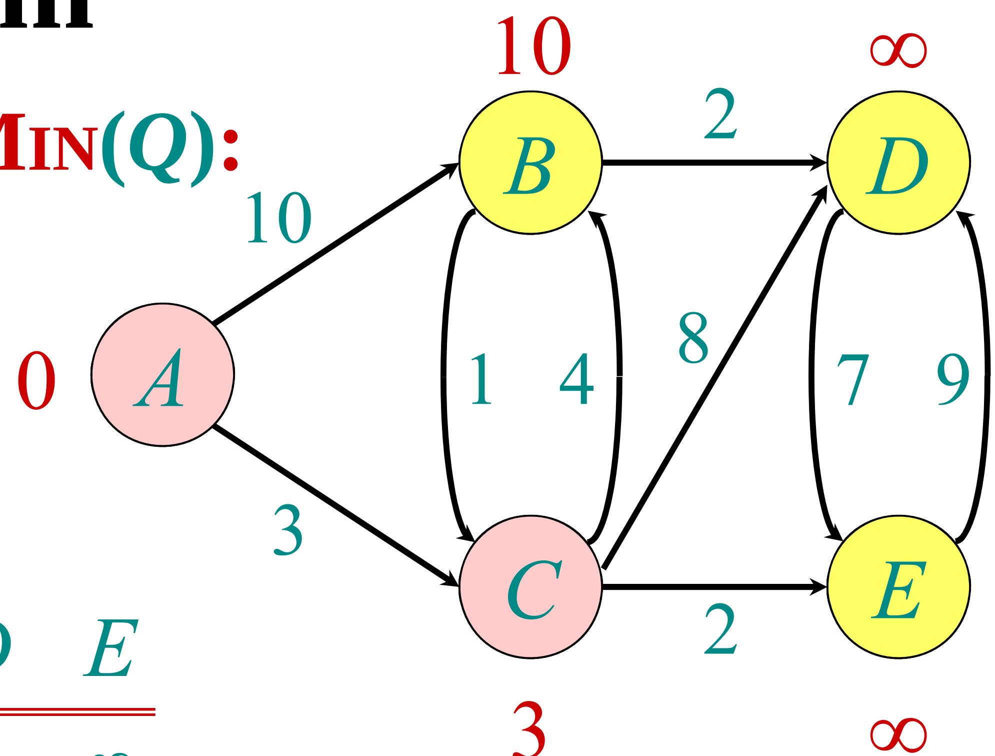
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	–	–

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

Example of Dijkstra's algorithm

"C" ← EXTRACT-MIN(Q):

S: { A, C }



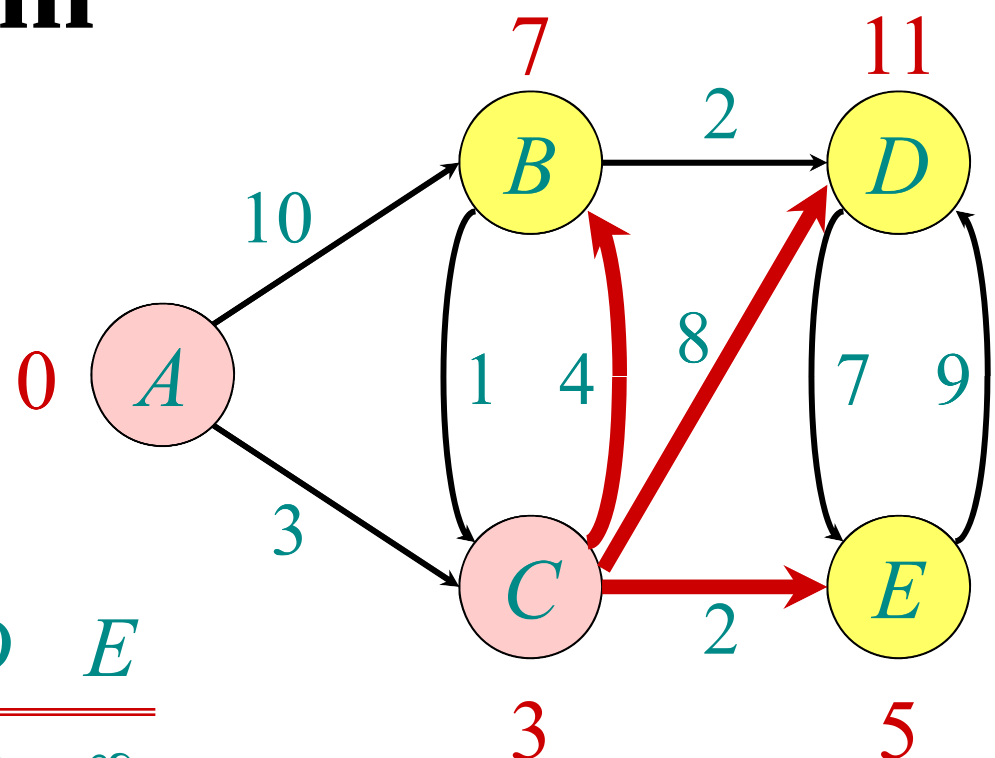
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	–	–

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

Example of Dijkstra's algorithm

Relax all edges leaving *C*:

S: { *A*, *C* }



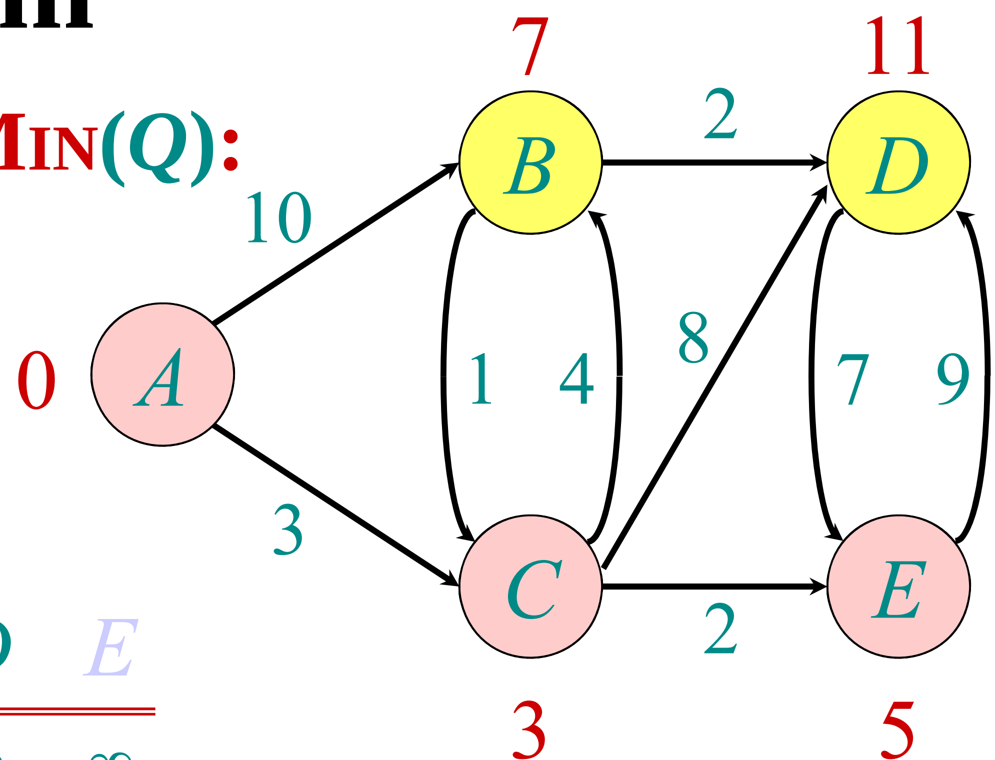
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	–	–
		7		11	5

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

Example of Dijkstra's algorithm

"*E*" ← EXTRACT-MIN(*Q*):

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



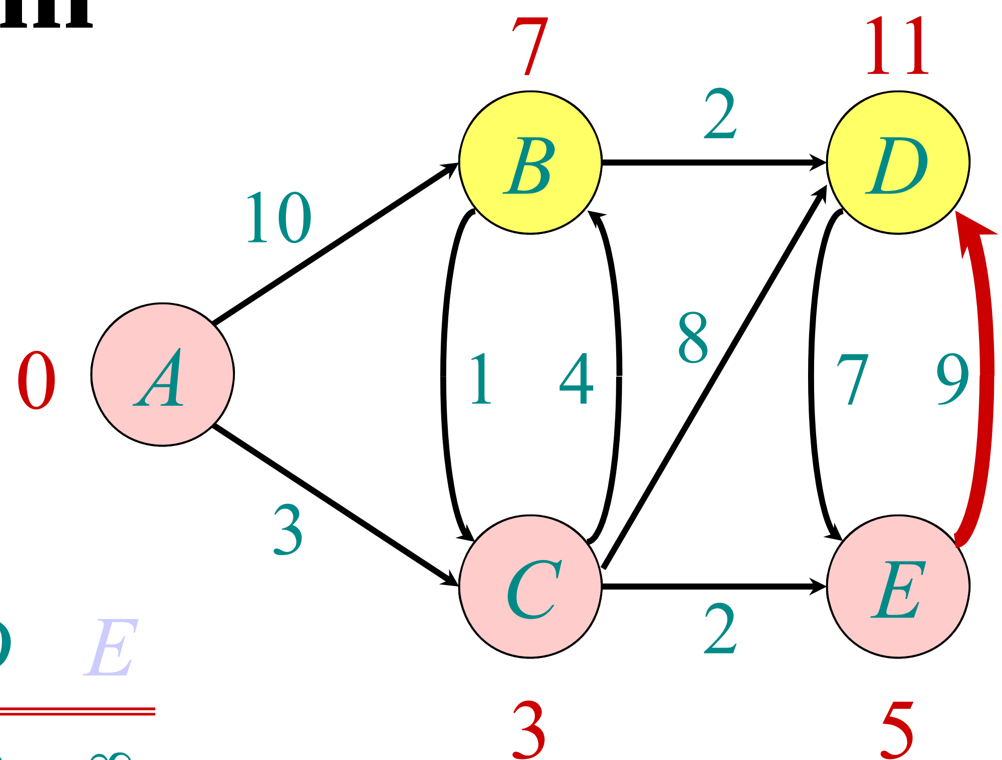
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	–	–
		7		11	5

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

Example of Dijkstra's algorithm

Relax all edges leaving *E*:

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

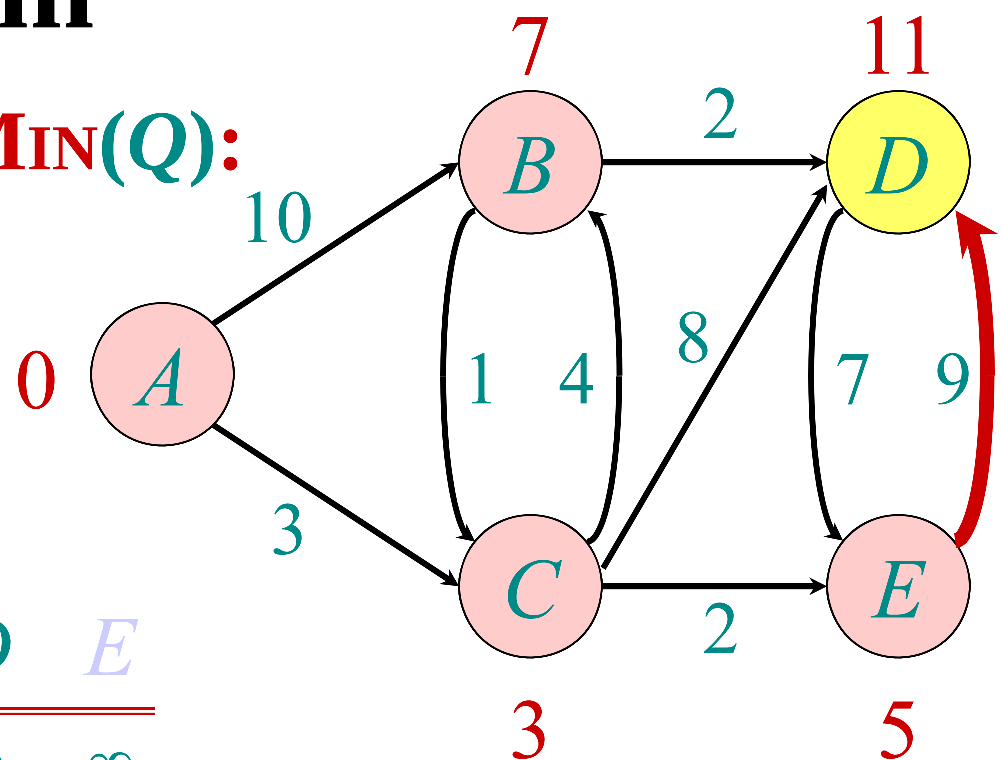


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

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

Example of Dijkstra's algorithm

"*B*" ← EXTRACT-MIN(*Q*):



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

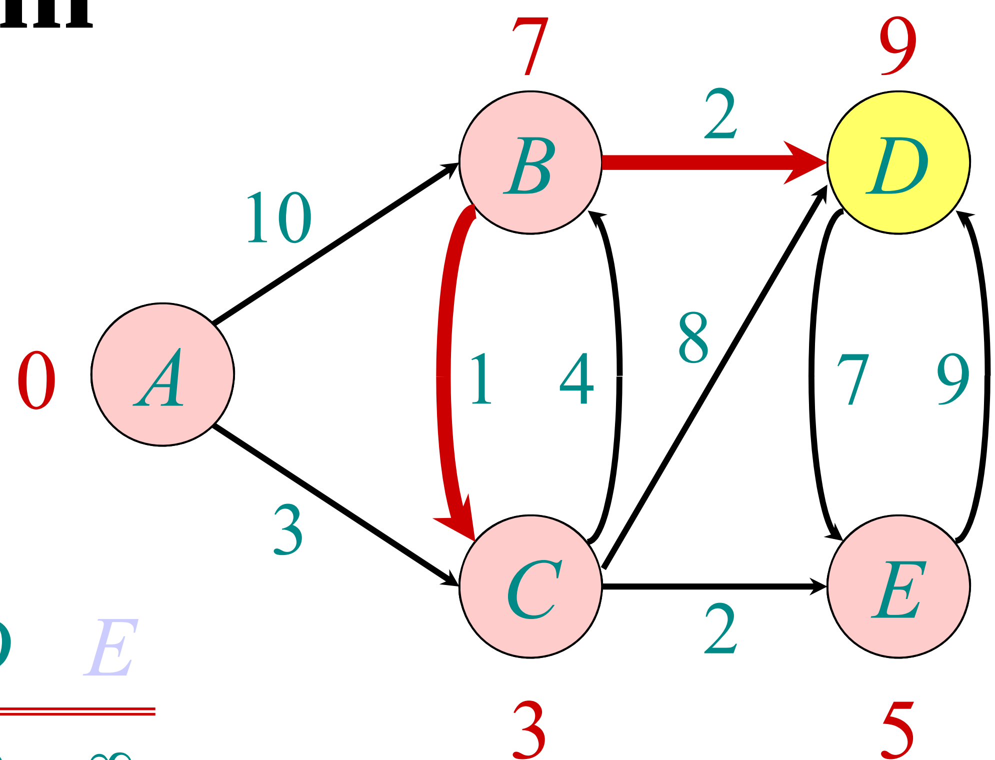
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

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

Example of Dijkstra's algorithm

Relax all edges leaving *B*:

S: { A, C, E, B }



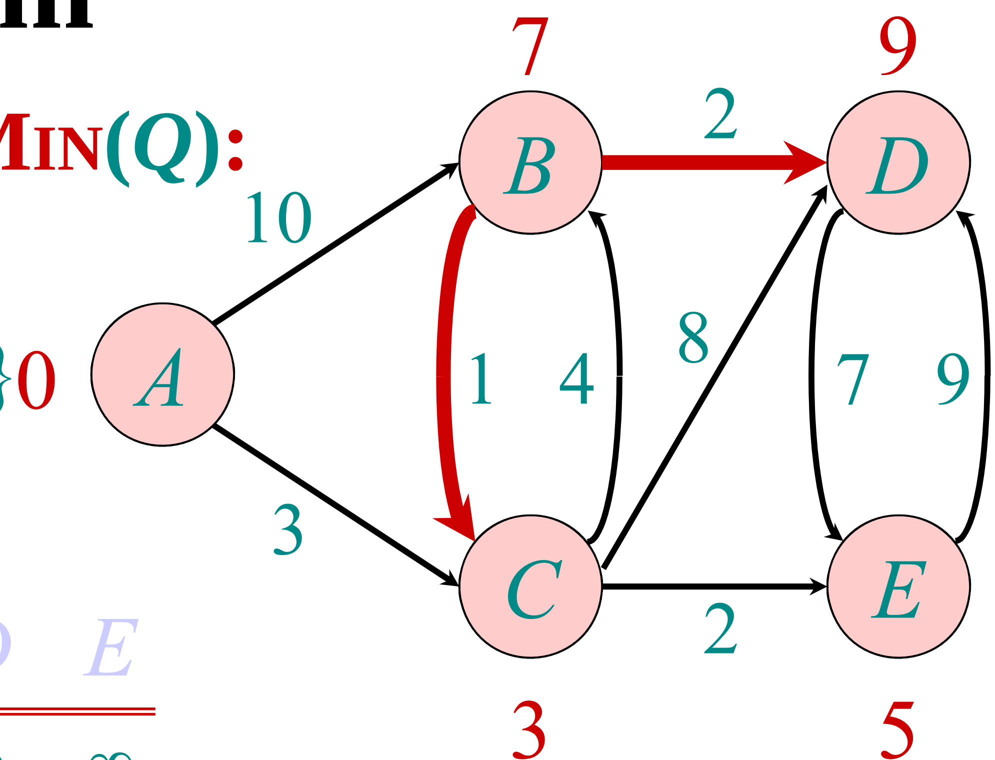
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

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

Example of Dijkstra's algorithm

"D" ← **EXTRACT-MIN(Q):**

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



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

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

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

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

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 $d[v] = \delta(s, v)$ 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.

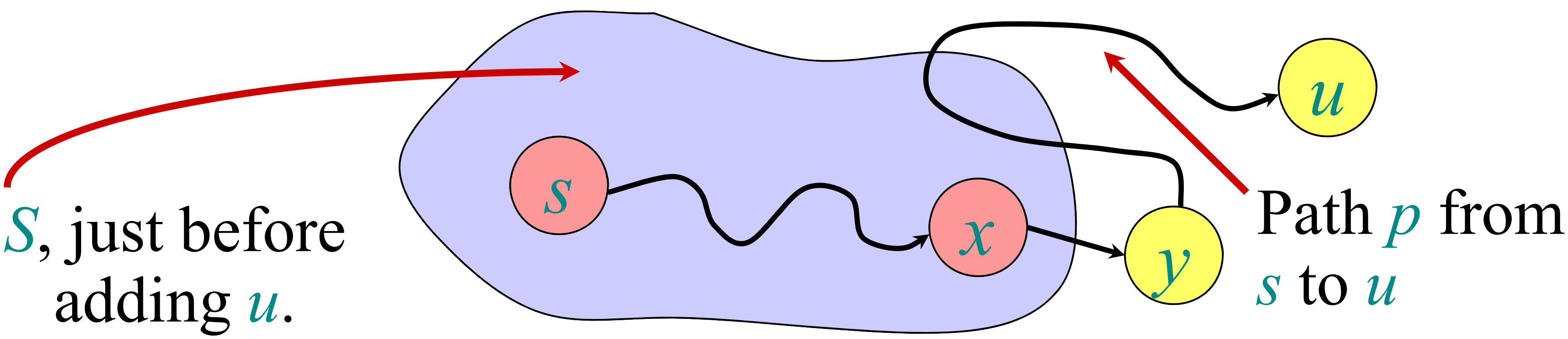
Proof. By induction.

- 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 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

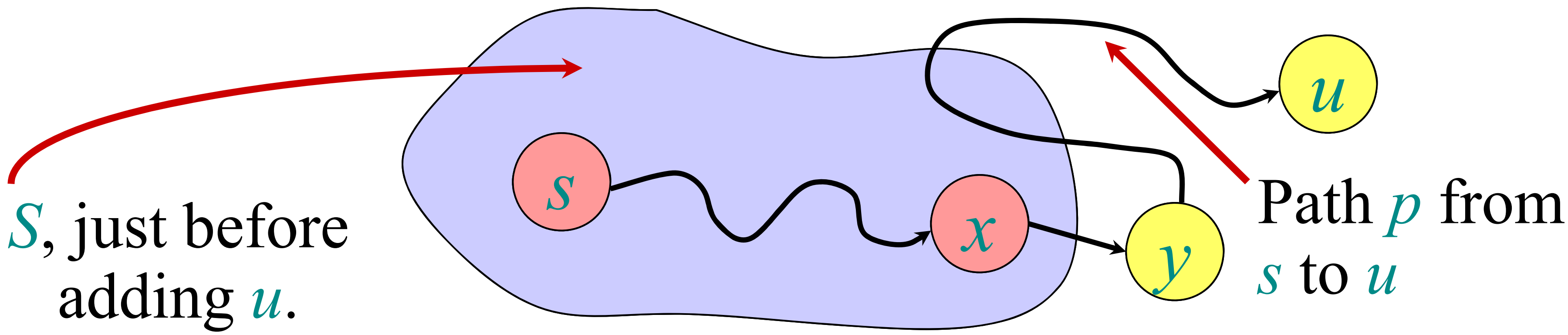
Theorem. (i) For all $v \in S$: $d[v] = \delta(s, v)$
(ii) For all $v \notin S$: $d[v]$ = weight of shortest path from s to v that uses only (besides v itself) vertices in S.

- (i) Need to show that $d[u] = \delta(s, u)$. Assume the contrary.
$\Rightarrow$ There is a path p from s to u with $w(p) < d[u]$. Because of (ii) that path uses vertices $\notin S$, in addition to u.
$\Rightarrow$ 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.

weights are nonnegative

assumption about path

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

```
while Q ≠ ∅ do
    u ← EXTRACT-MIN(Q)
    S ← S ∪ {u}
    for each v ∈ Adj[u] do
        if d[v] > d[u] + w(u, v) then
            d[v] ← d[u] + w(u, v)
```

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

Correctness of BFS

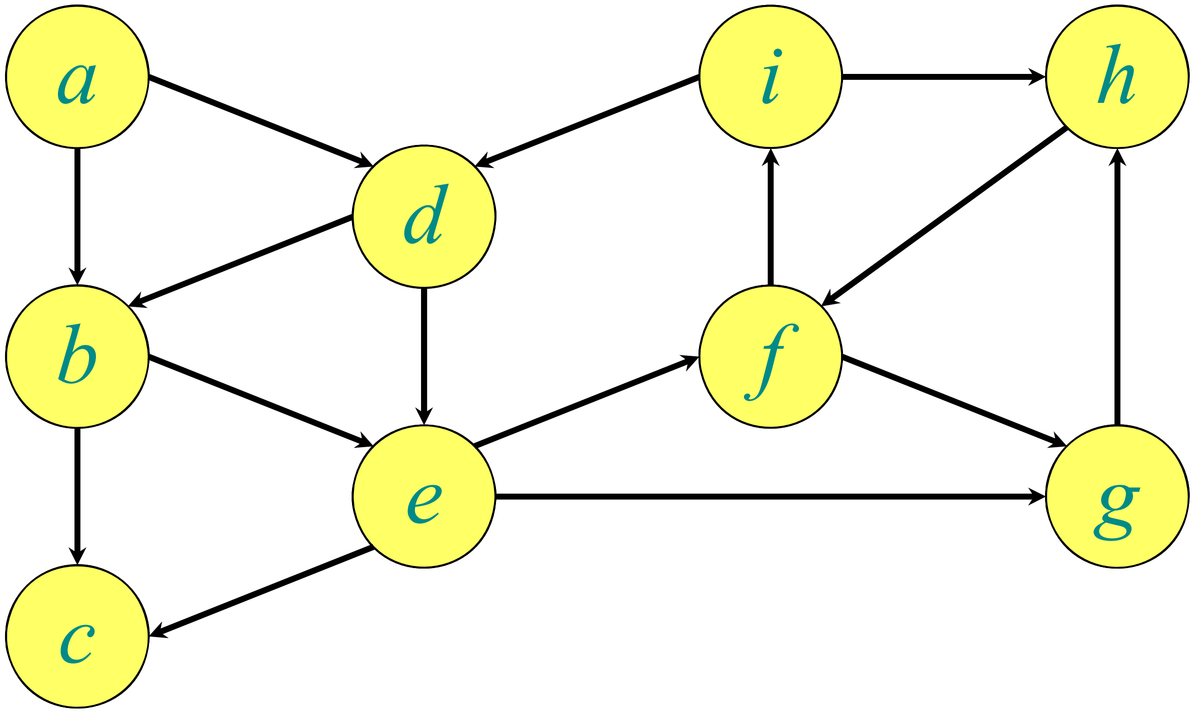
while $Q \neq \emptyset$
  do $u \leftarrow$ DEQUEUE(Q)
    for each $v \in Adj[u]$
      do if $d[v] = \infty$
        then $d[v] \leftarrow d[u] + 1$
          ENQUEUE(Q, v)

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 $d[v] = d[u] + 1$.

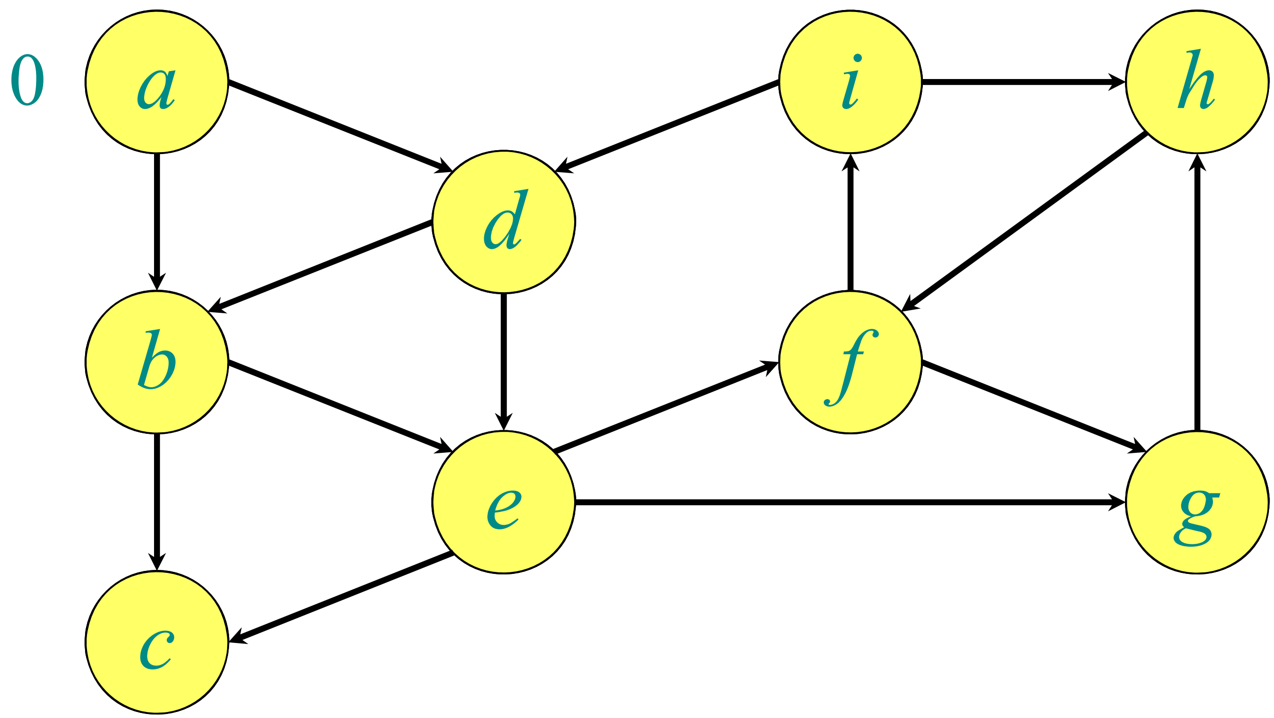
Example of breadth-first search



Q:

$d[v]$

Example of breadth-first search

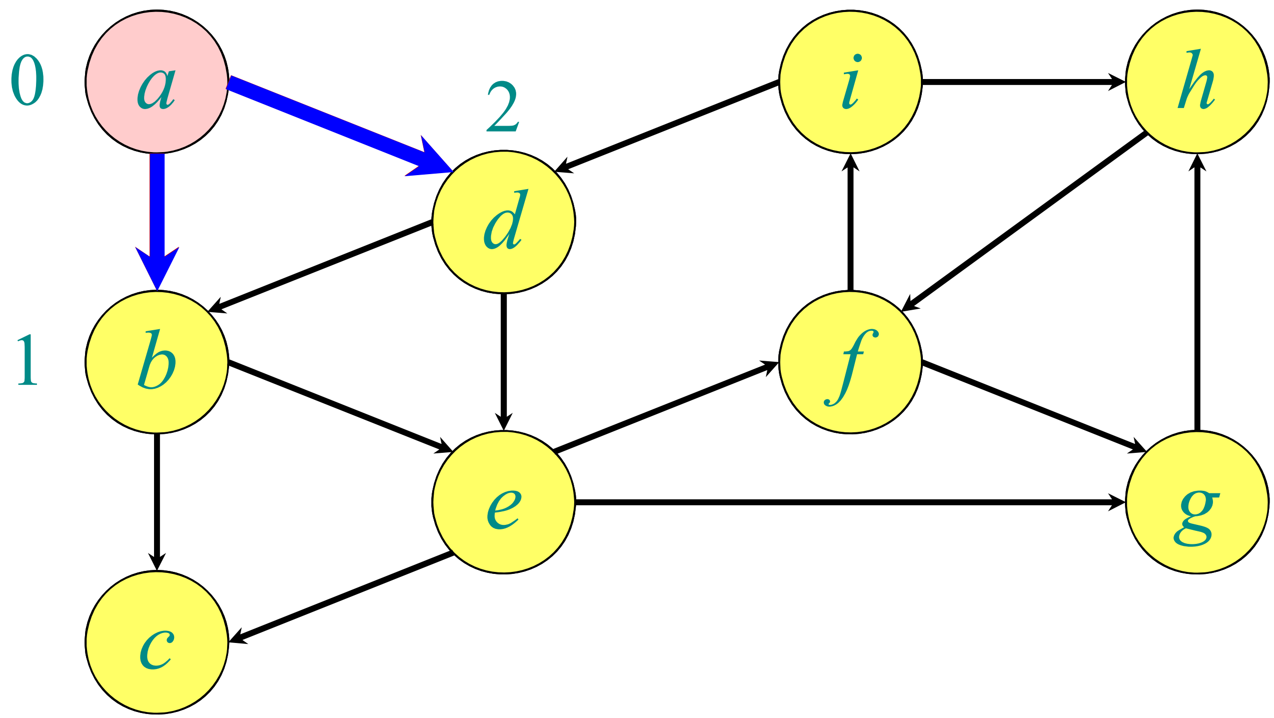


0

Q: *a*

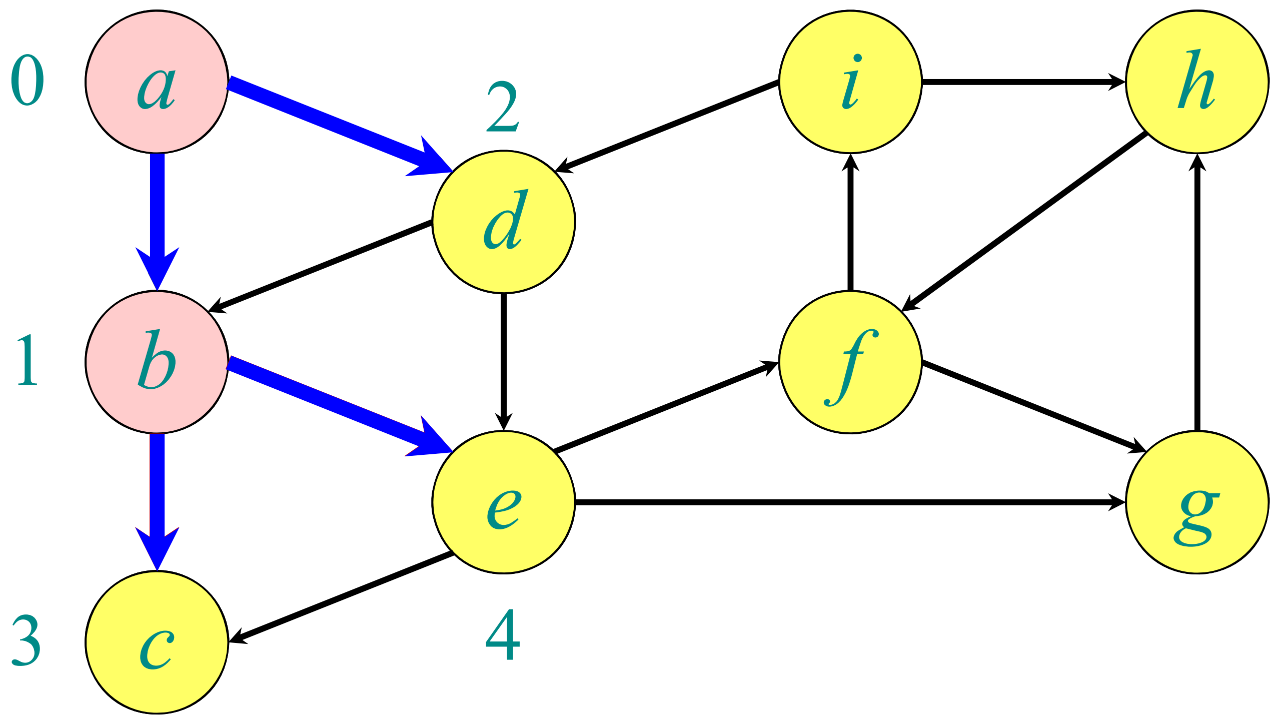
d[*v*] 0

Example of breadth-first search



		1	2
Q:	*a*	*b*	*d*
d[*v*]	0	1	1

Example of breadth-first search

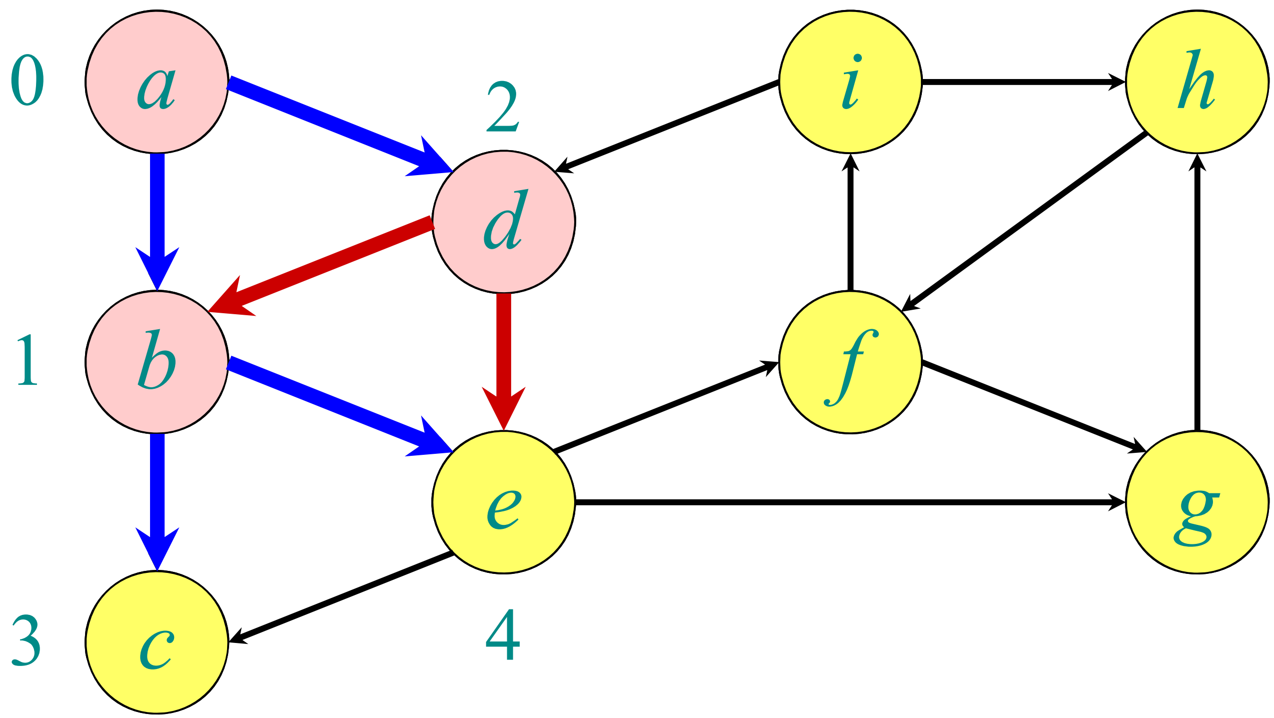


2 3 4

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

d[*v*] 0 1 1 2 2

Example of breadth-first search

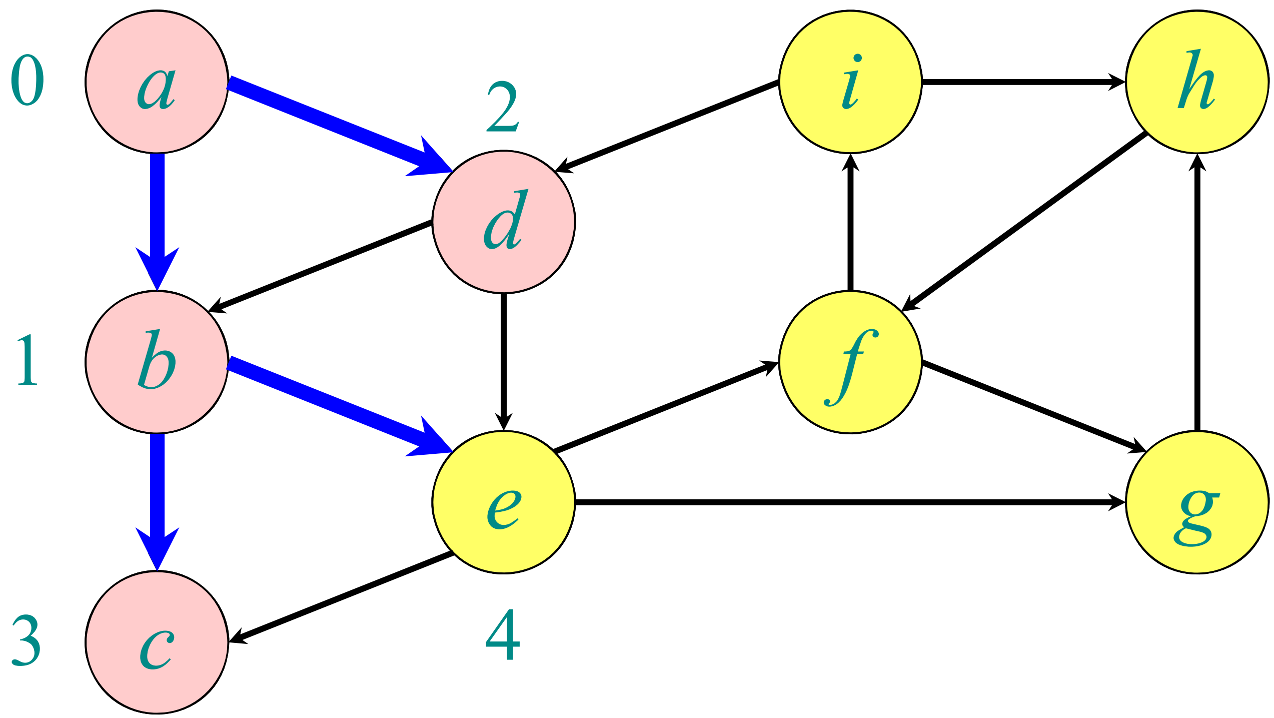


3 4

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

d[*v*] 0 1 1 2 2

Example of breadth-first search

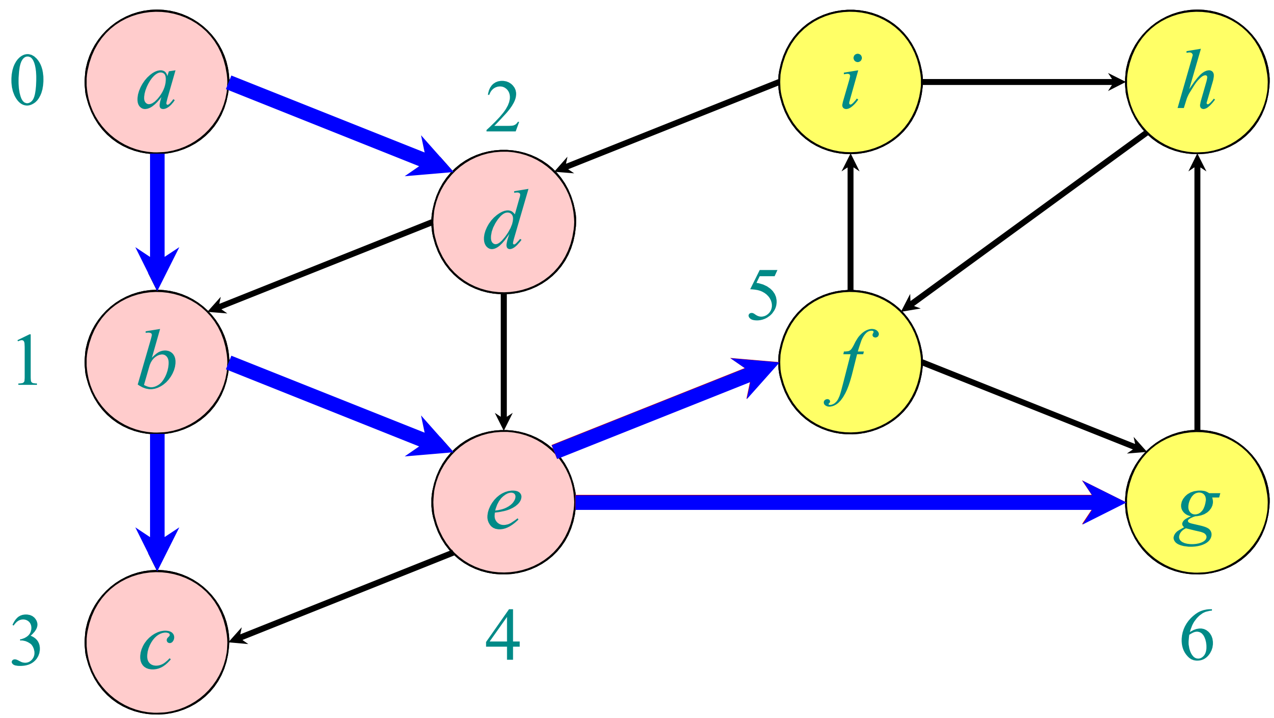


4

Q: *a b d c e*

d[*v*] 0 1 1 2 2

Example of breadth-first search

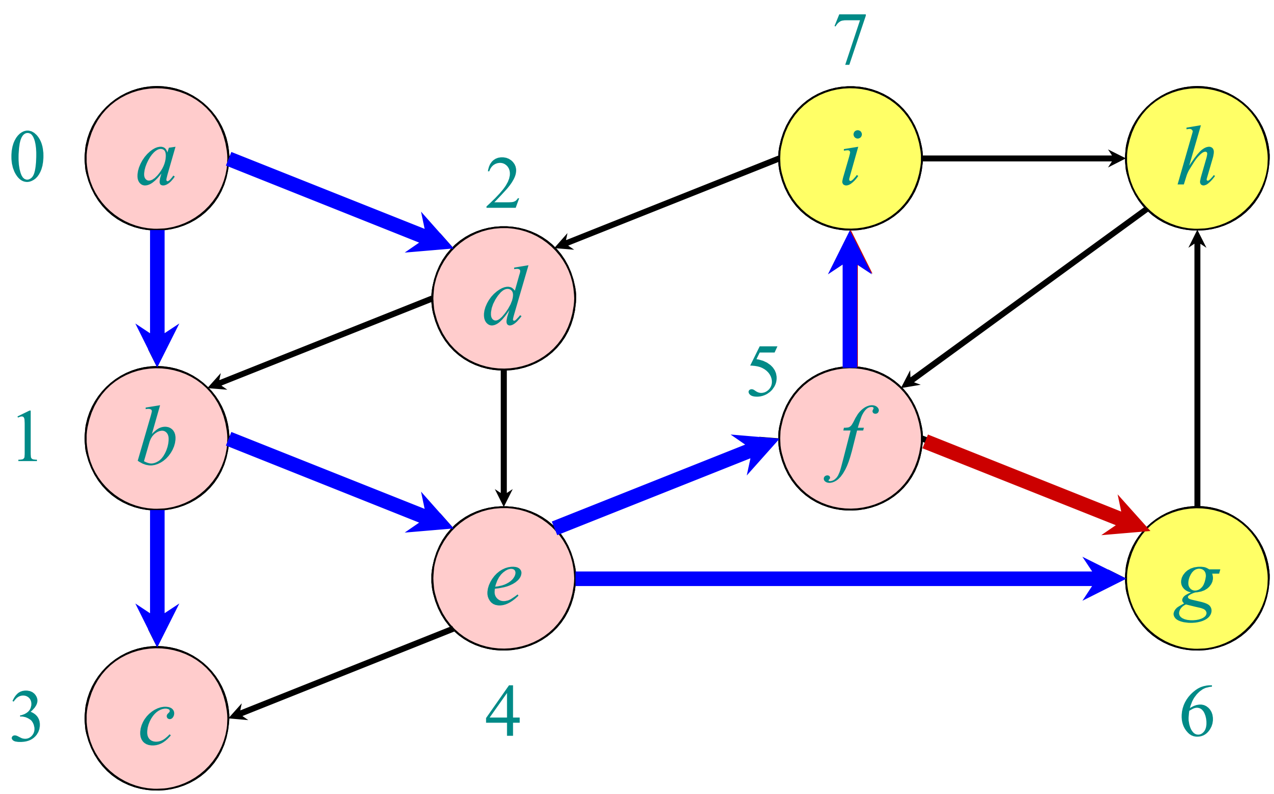


5 6

Q: *a b d c e f g*

d[*v*] 0 1 1 2 2 3 3

Example of breadth-first search

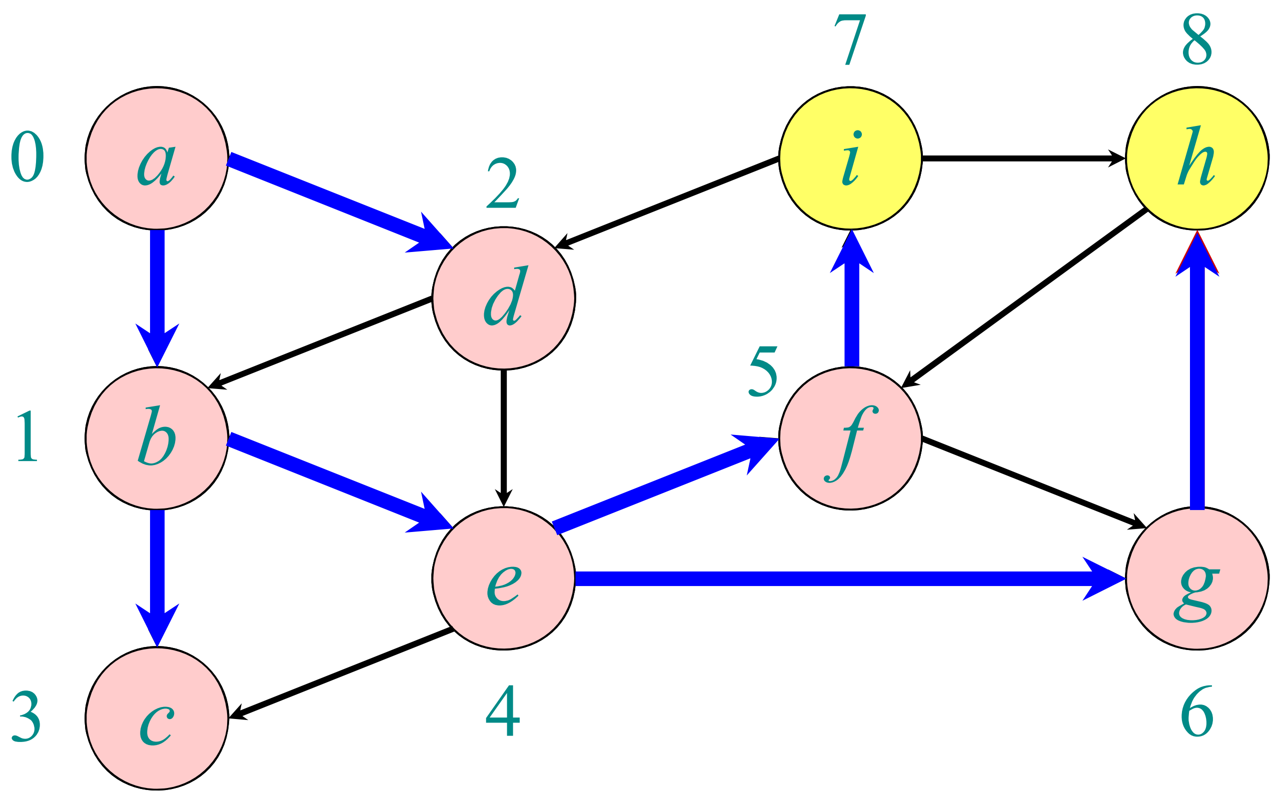


6 7

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

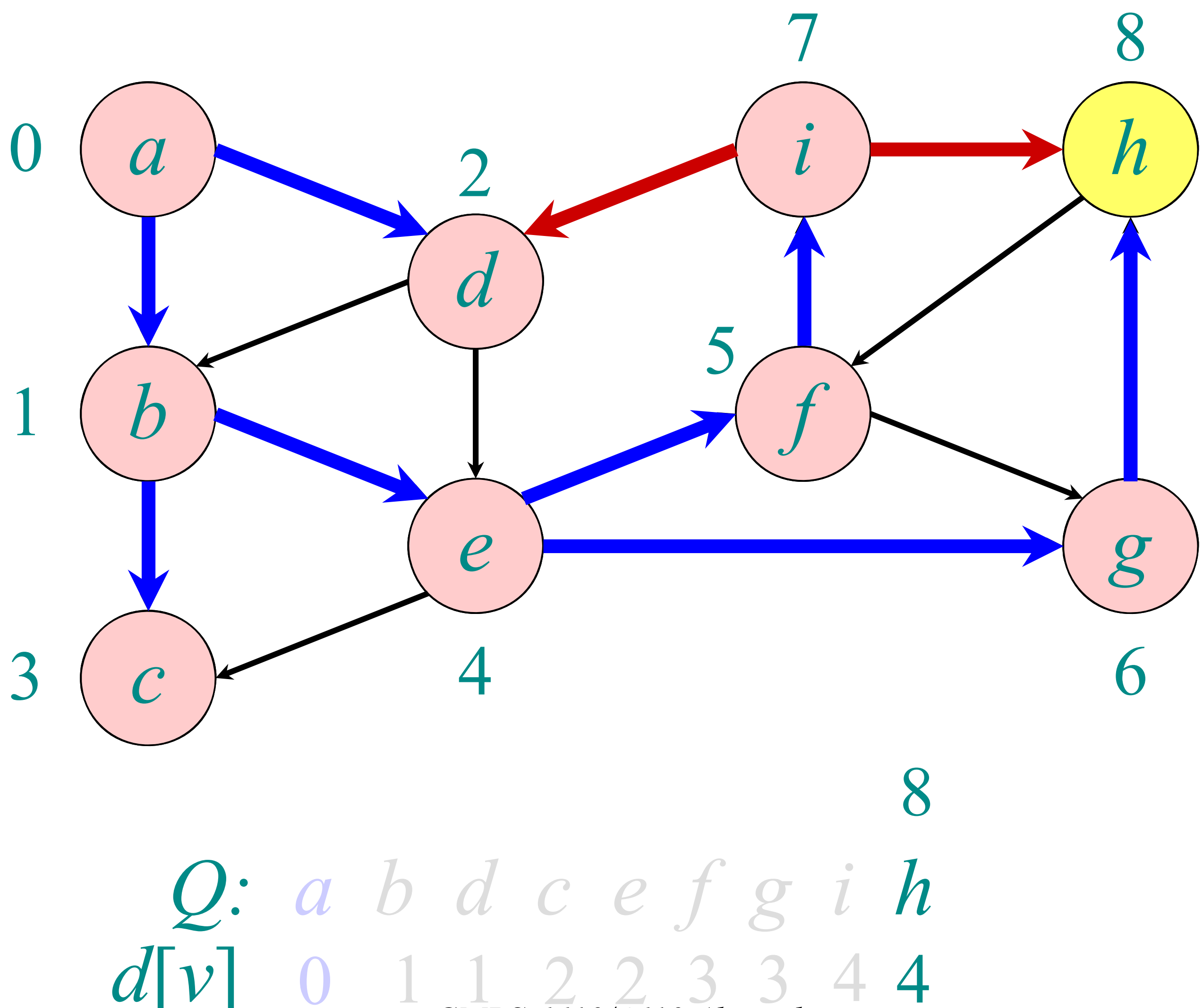
d[*v*] 0 1 1 2 2 3 3 4

Example of breadth-first search

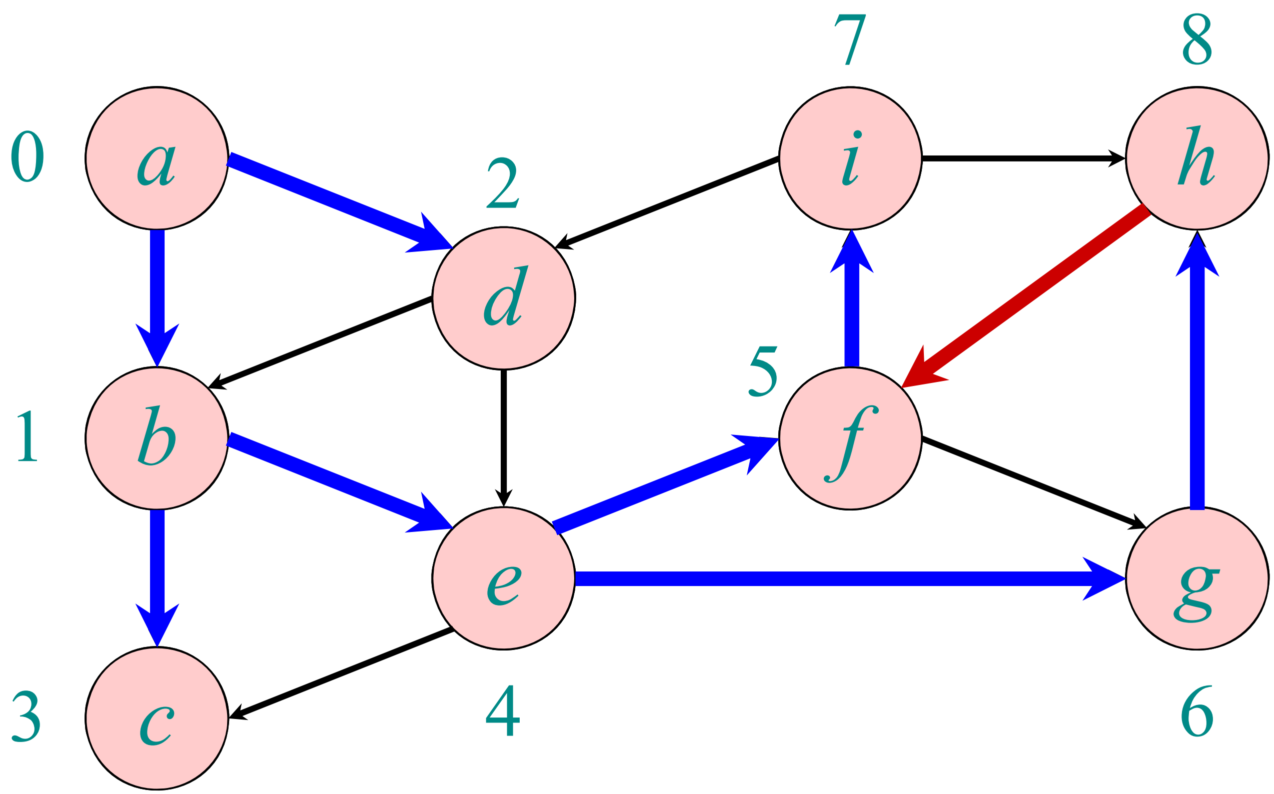


								7	8
Q:	*a*	*b*	*d*	*c*	*e*	*f*	*g*	*i*	*h*
$d[v]$	0	1	1	2	2	3	3	4	4

Example of breadth-first search



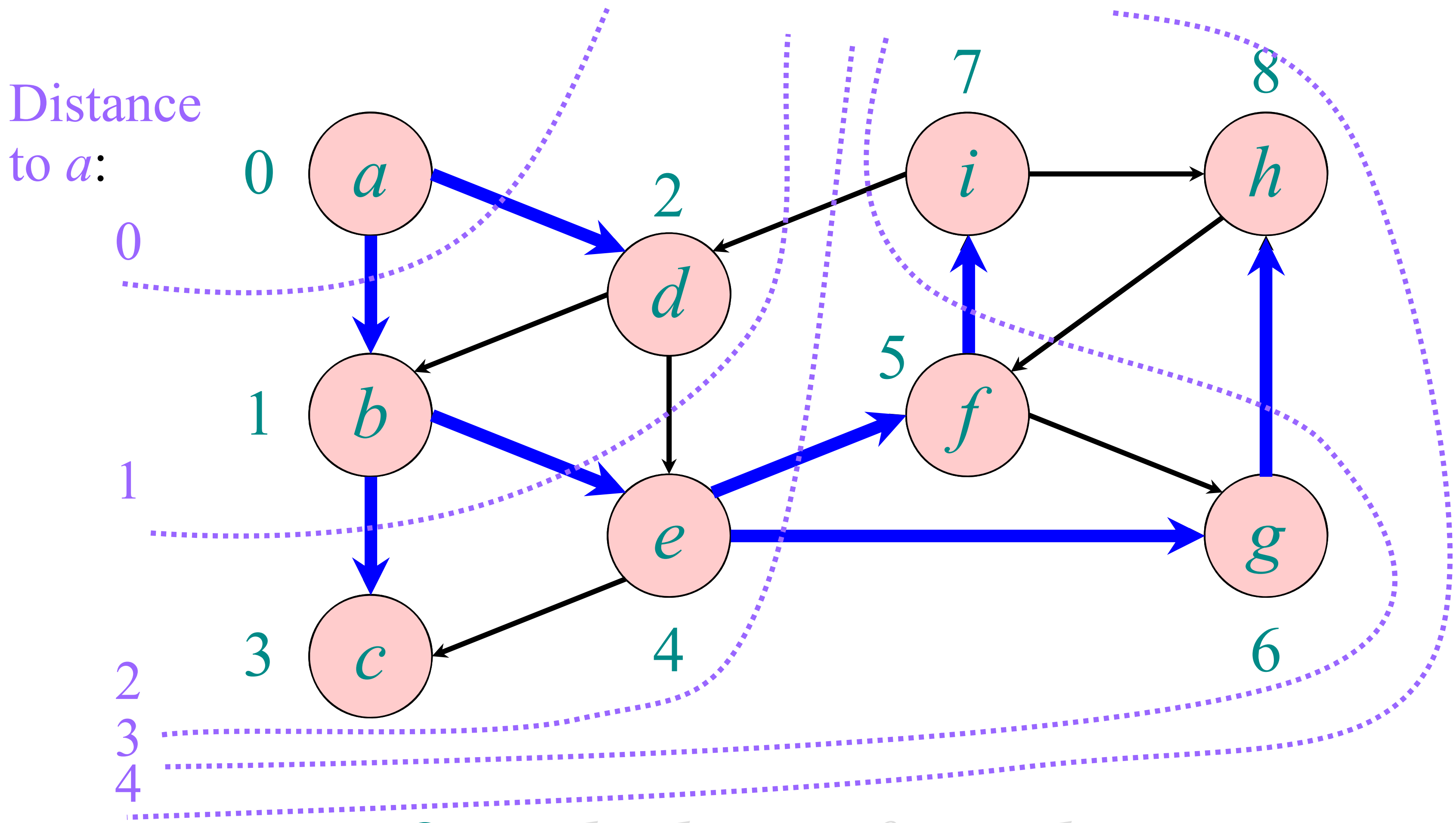
Example of breadth-first search



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

d[*v*] 0 1 1 2 2 3 3 4 4

Example of breadth-first search



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

d[*v*] 0 1 1 2 2 3 3 4 4

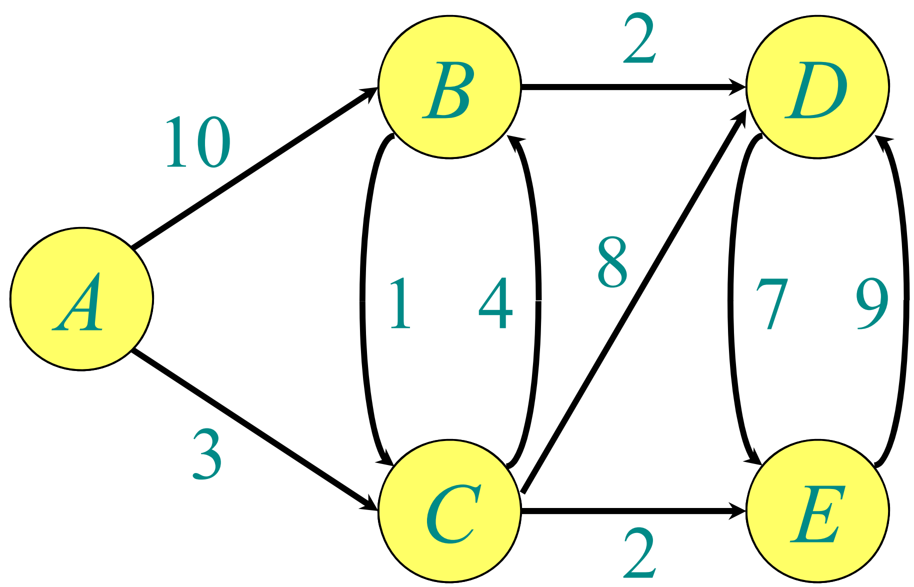
How to find the actual shortest paths?

Store a predecessor tree:

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
  do $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$ ⊳ 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:

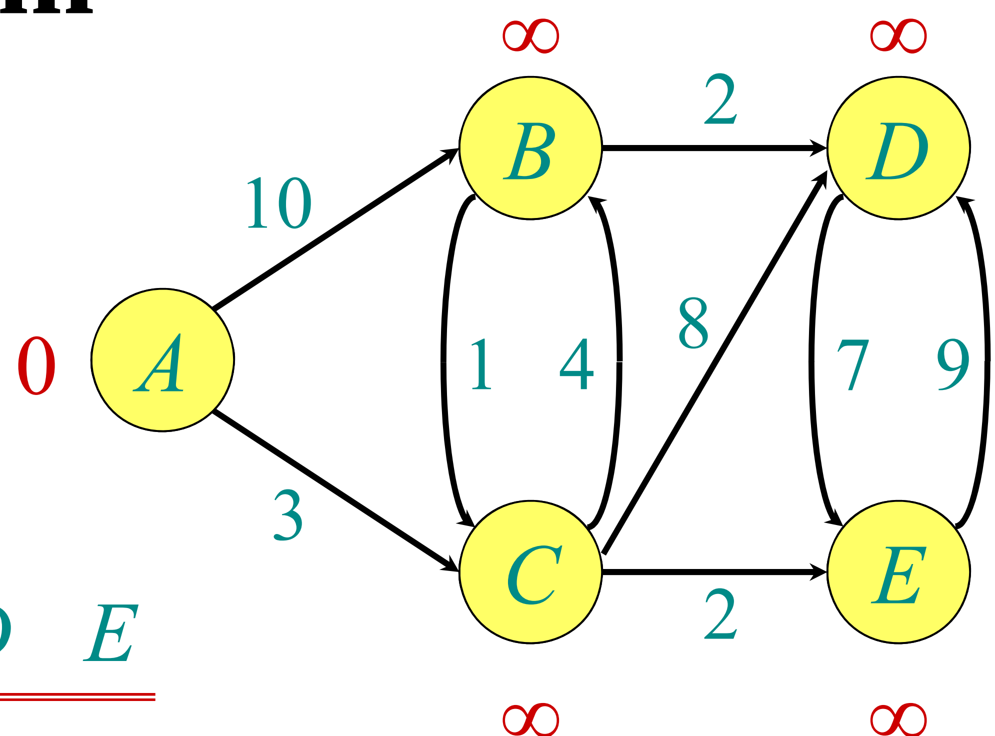



```
while Q ≠ ∅ do
    u ← EXTRACT-MIN(Q)
    S ← S ∪ {u}
    for each v ∈ Adj[u] do
        if d[v] > d[u] + w(u, v) then
            d[v] ← d[u] + w(u, v)
            π[v] ← u
```

Example of Dijkstra's algorithm

Initialize:

S: {}

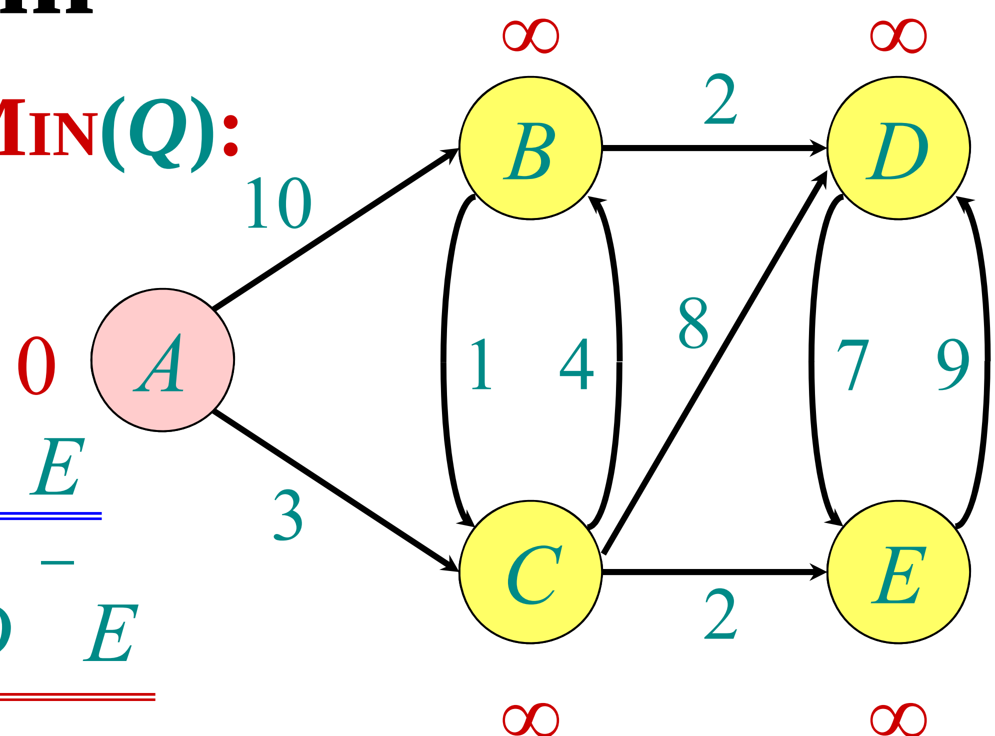


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞

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

Example of Dijkstra's algorithm

"*A*" ← EXTRACT-MIN(*Q*):



S: { *A* }

π:	*A*	*B*	*C*	*D*	*E*
	–	–	–	–	–

Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞

```
while Q ≠ ∅ do
   u ← EXTRACT-MIN(Q)
   S ← S ∪ {u}
   for each v ∈ Adj[u] do
      if d[v] > d[u] + w(u, v) then
         d[v] ← d[u] + w(u, v)
         π[v] ← u
```

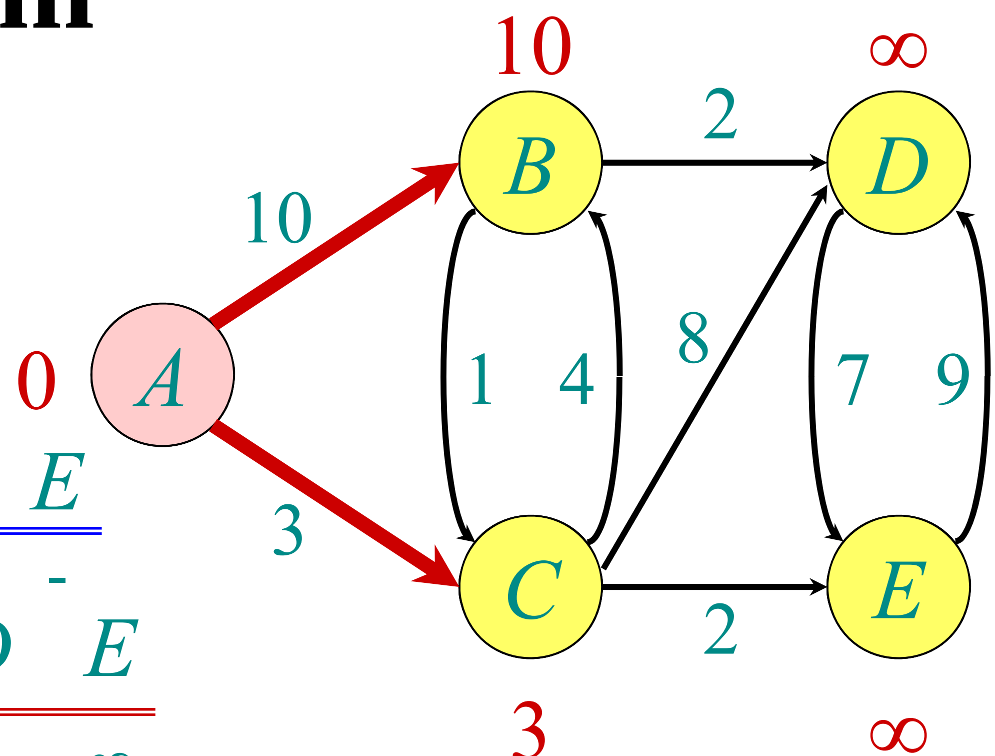
Example of Dijkstra's algorithm

Relax all edges leaving *A*:

S: { *A* }

π:	*A*	*B*	*C*	*D*	*E*
	–	–	–	–	-

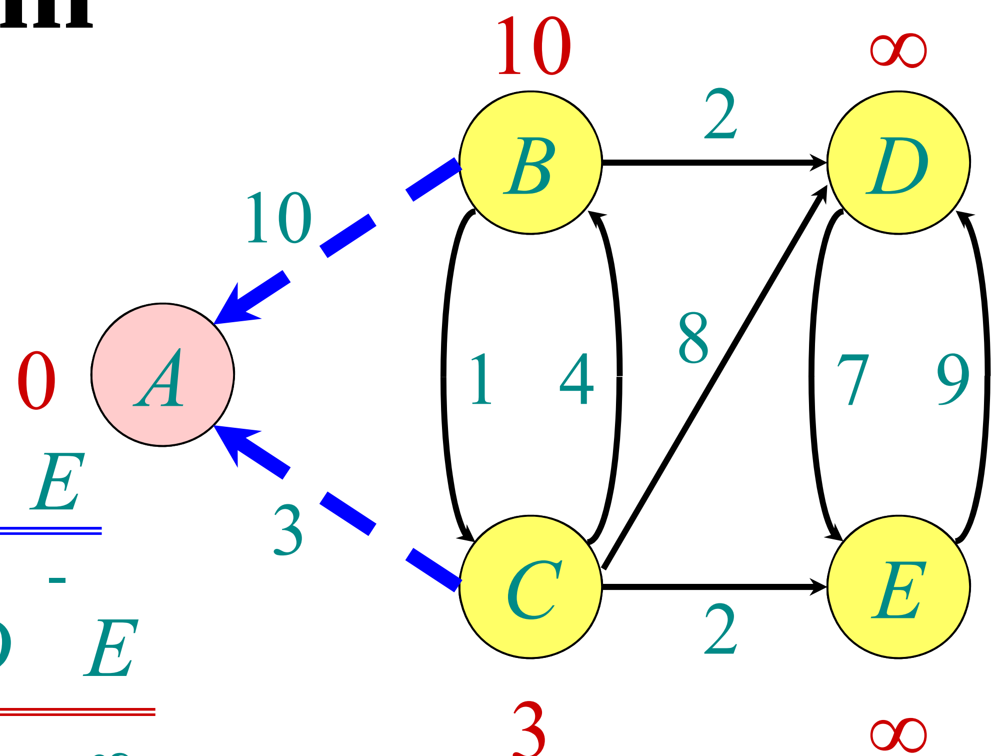
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	–	–




```
while Q ≠ ∅ do
    u ← EXTRACT-MIN(Q)
    S ← S ∪ {u}
    for each v ∈ Adj[u] do
        if d[v] > d[u] + w(u, v) then
            d[v] ← d[u] + w(u, v)
            π[v] ← u
```

Example of Dijkstra's algorithm

Relax all edges leaving *A*:



S: { *A* }

π:

A	*B*	*C*	*D*	*E*
–	A	A	–	-

Q:

A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞
	10	3	–	–

```
while Q ≠ ∅ do
    u ← EXTRACT-MIN(Q)
    S ← S ∪ {u}
    for each v ∈ Adj[u] do
        if d[v] > d[u] + w(u, v) then
            d[v] ← d[u] + w(u, v)
            π[v] ← u
```

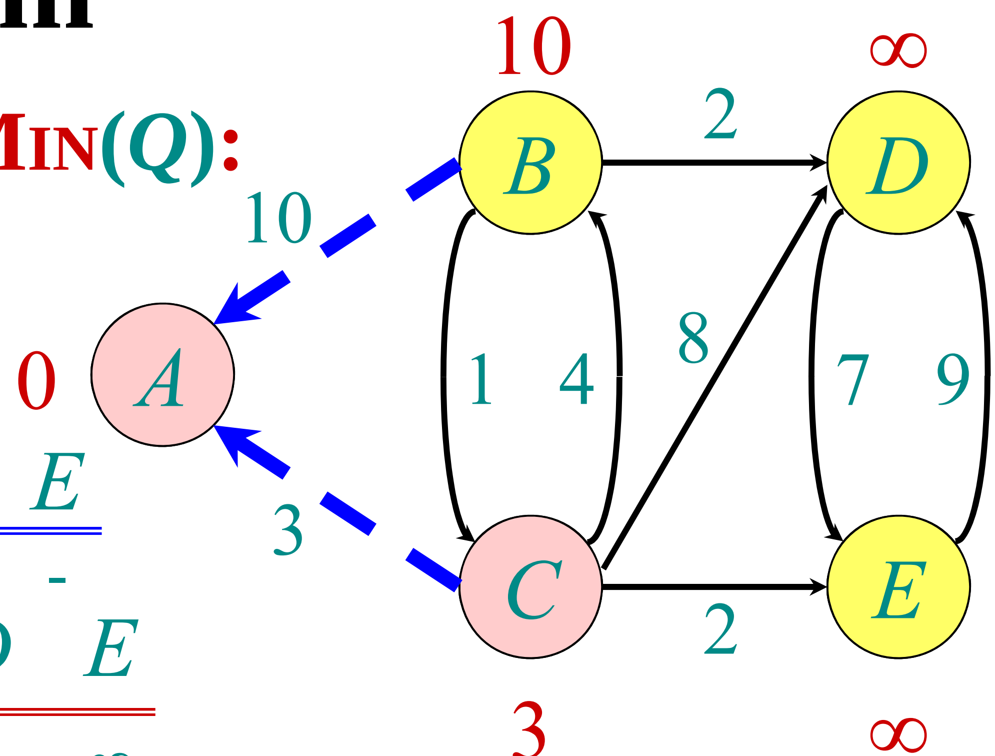
Example of Dijkstra's algorithm

"*C*" ← EXTRACT-MIN(*Q*):

S: { *A*, *C* }

π:	*A*	*B*	*C*	*D*	*E*
	–	A	A	–	-

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



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

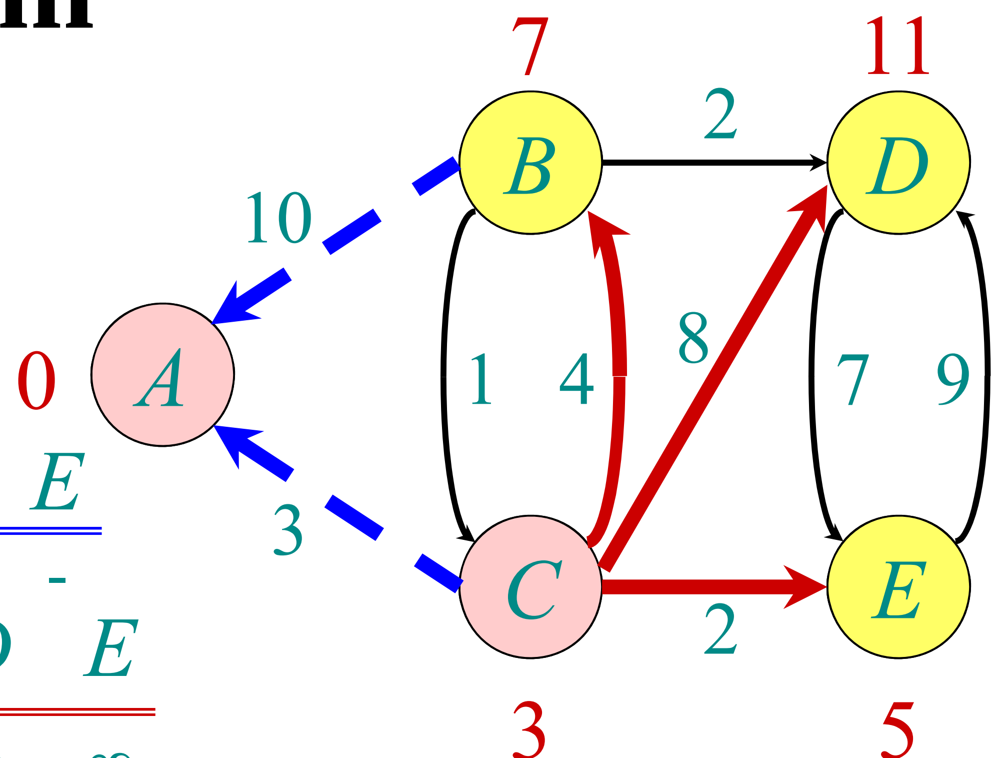
Example of Dijkstra's algorithm

Relax all edges leaving *C*:

S: { *A*, *C* }

π:	*A*	*B*	*C*	*D*	*E*
	–	A	A	–	-

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




```
while Q ≠ ∅ do
   u ← EXTRACT-MIN(Q)
   S ← S ∪ {u}
   for each v ∈ Adj[u] do
      if d[v] > d[u] + w(u, v) then
         d[v] ← d[u] + w(u, v)
         π[v] ← u
```

Example of Dijkstra's algorithm

Relax all edges leaving *C*:

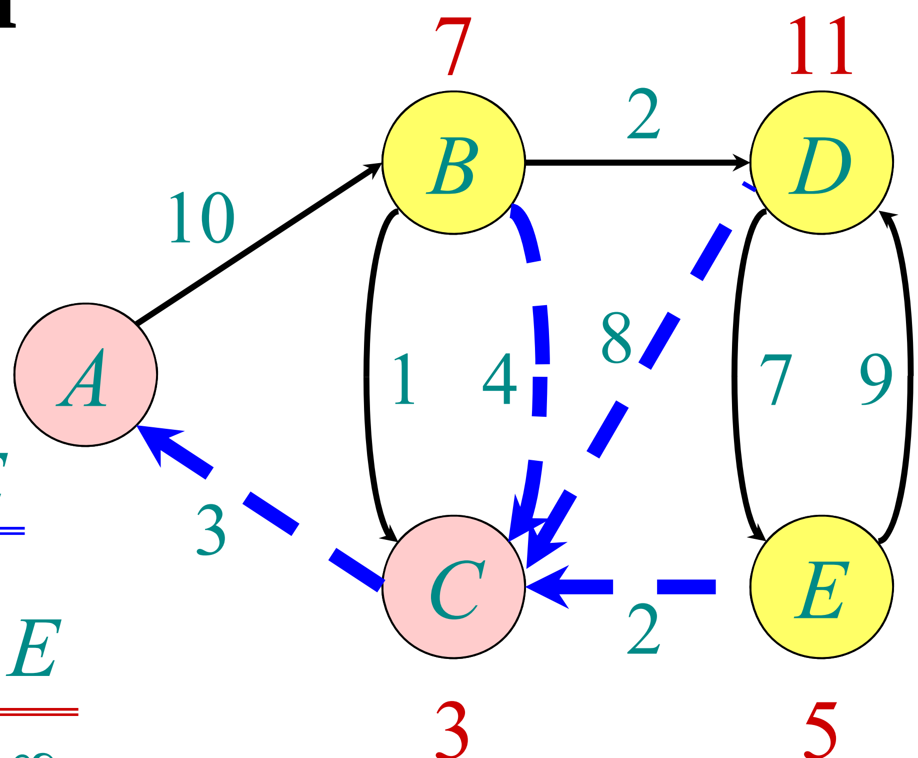
S: { *A*, *C* }

π:

A	*B*	*C*	*D*	*E*
–	A	A	–	-

Q:

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



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

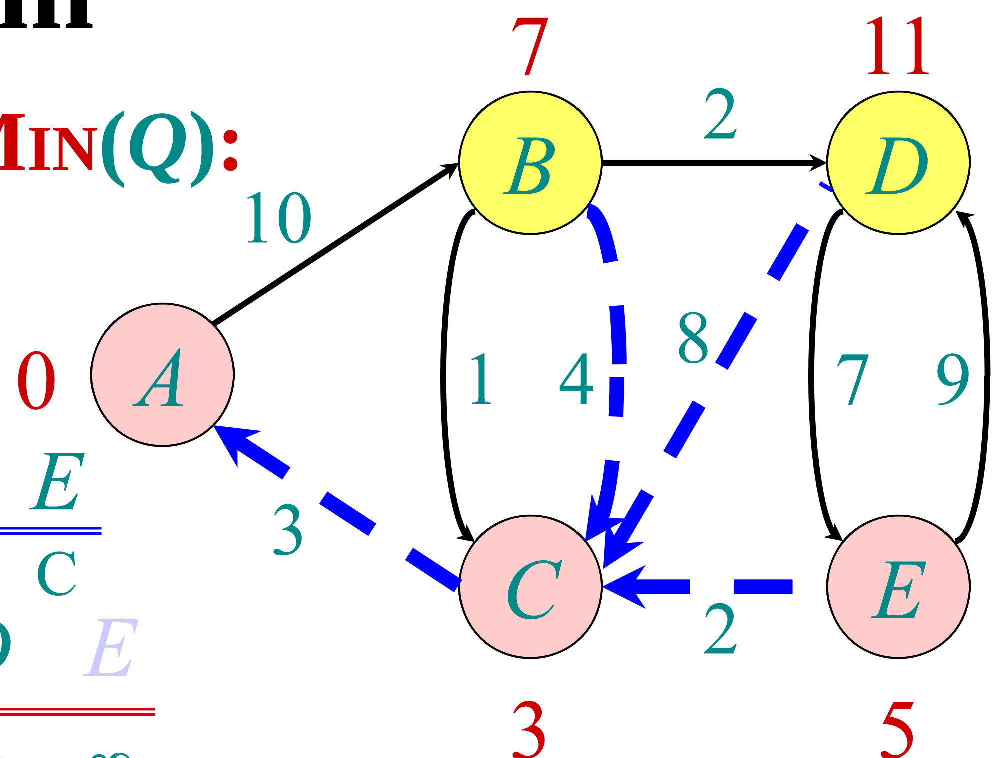
Example of Dijkstra's algorithm

"E" ← EXTRACT-MIN(*Q*):

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

π:	*A*	*B*	*C*	*D*	*E*
	–	C	A	C	C

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



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

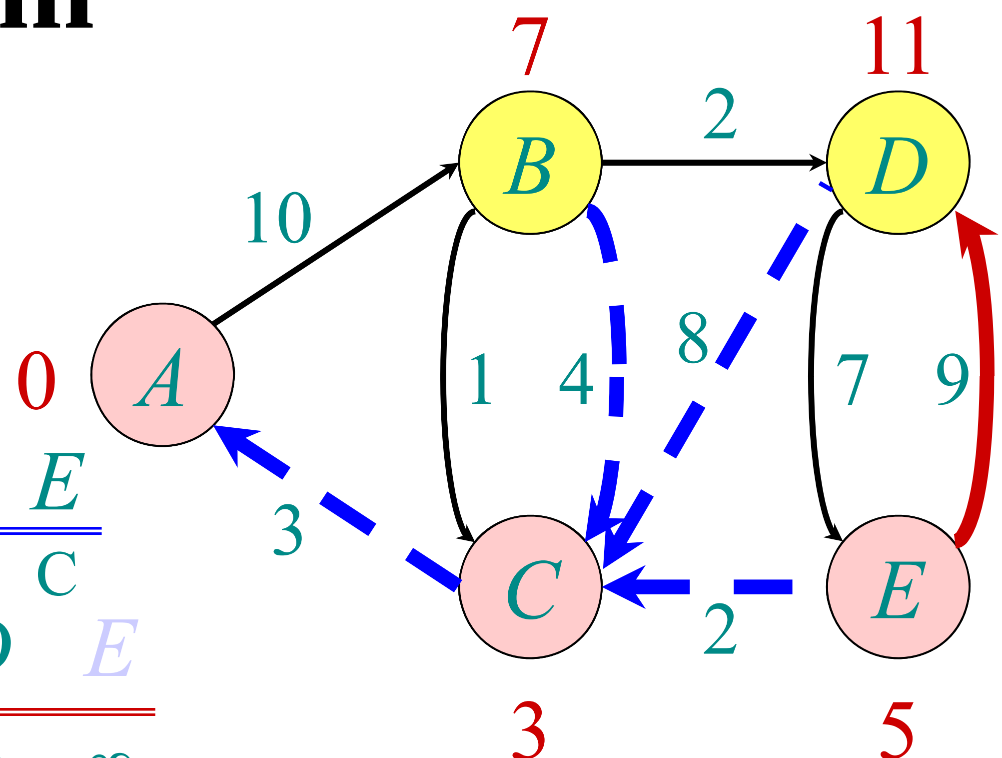
Example of Dijkstra's algorithm

Relax all edges leaving *E*:

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

π:	*A*	*B*	*C*	*D*	*E*
	–	C	A	C	C

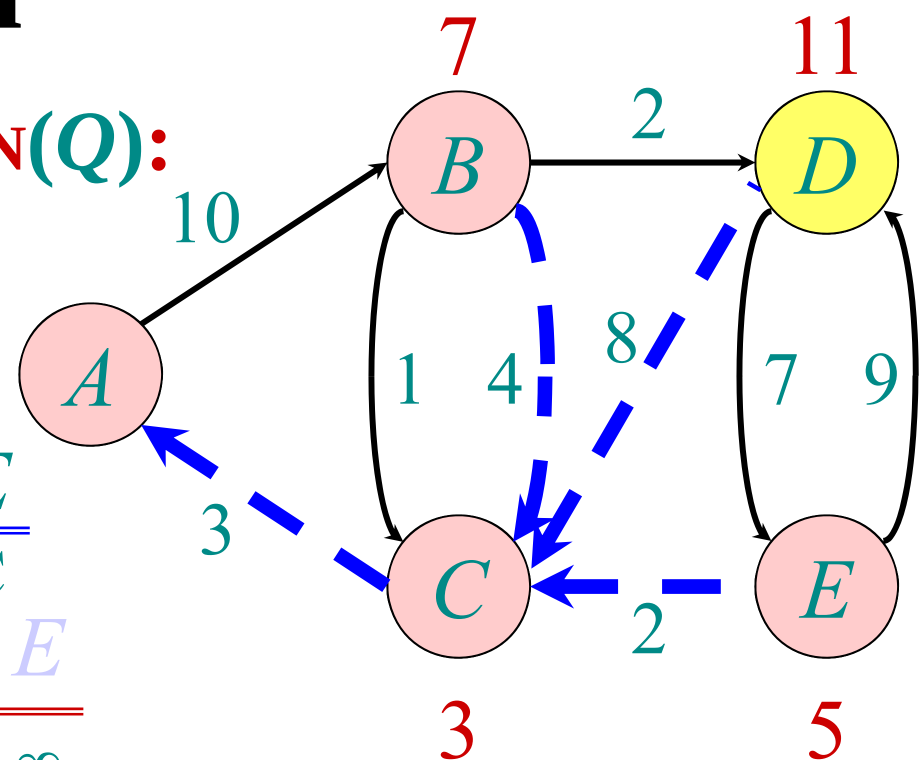
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	



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

Example of Dijkstra's algorithm

"*B*" ← EXTRACT-MIN(*Q*):



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

π:	*A*	*B*	*C*	*D*	*E*
	–	C	A	C	C

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

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

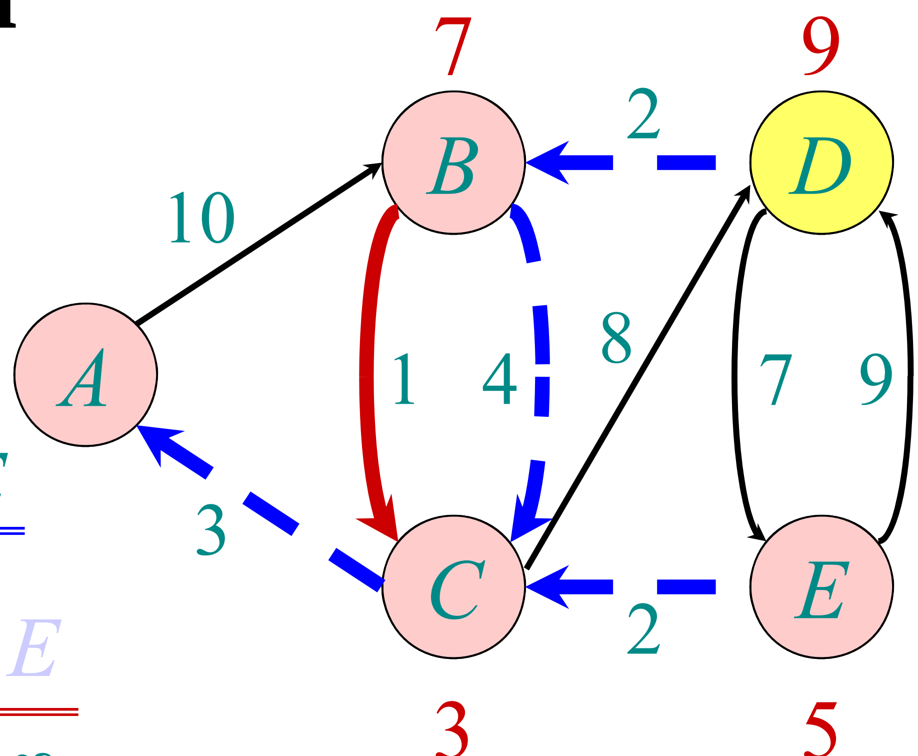
Example of Dijkstra's algorithm

Relax all edges leaving *B*:

S: { A, C, E, B }

π:	A	B	C	D	E
	–	C	A	B	C

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



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

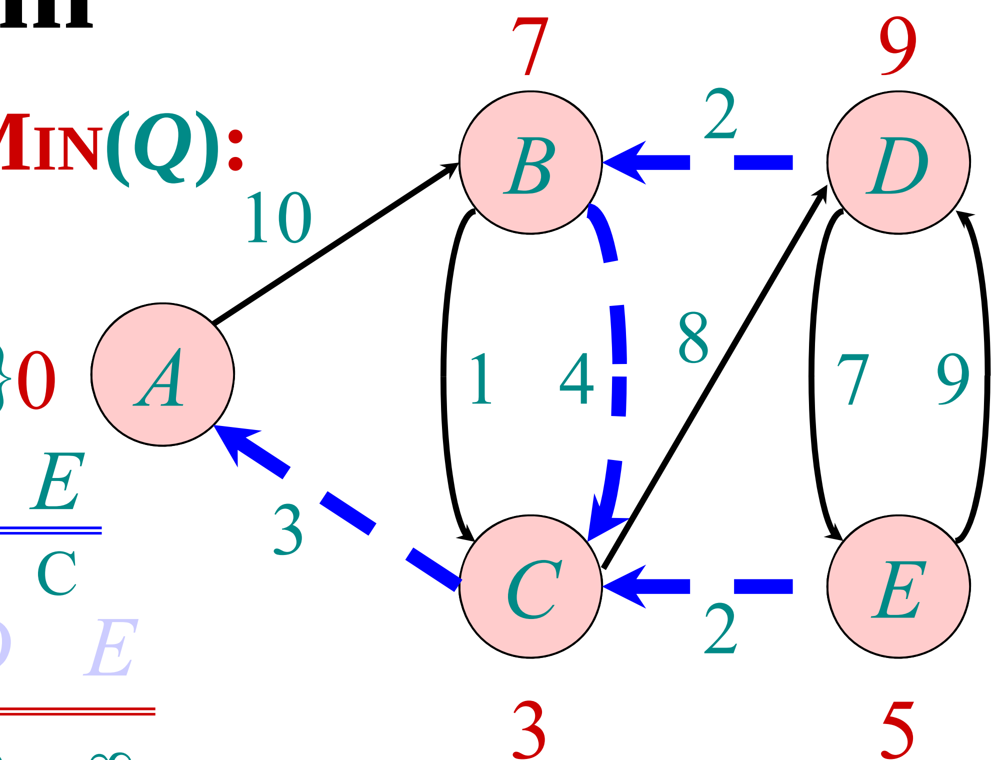
Example of Dijkstra's algorithm

"D" ← EXTRACT-MIN(Q):

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

π:	A	B	C	D	E
	–	C	A	B	C

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

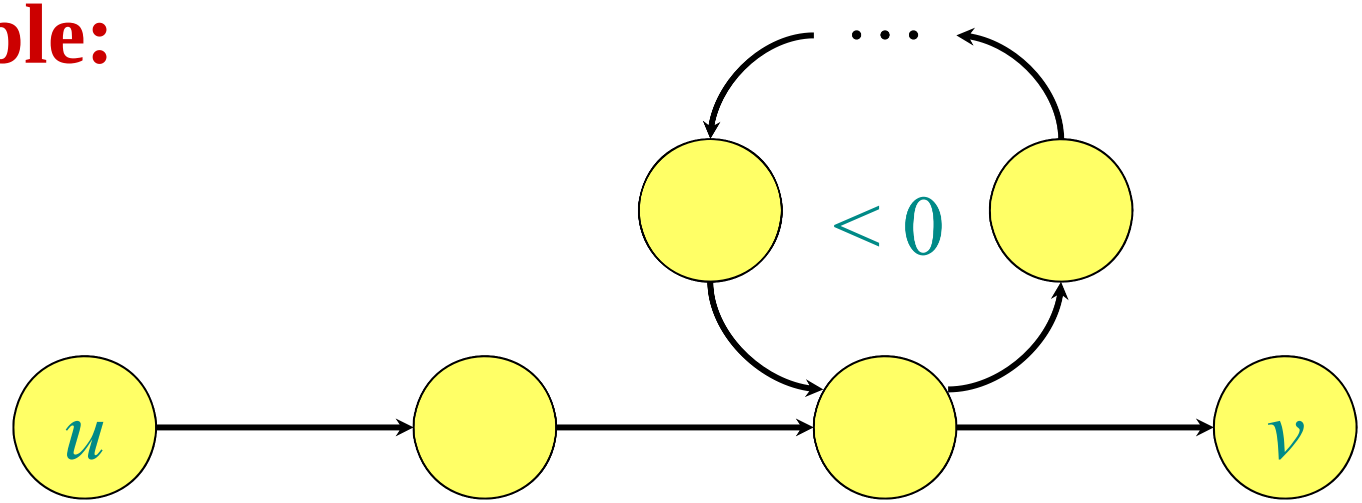


while $Q \neq \emptyset$ **do**
  $u \leftarrow$ EXTRACT-MIN(Q)
  $S \leftarrow S \cup \{u\}$
  for each $v \in Adj[u]$ **do**
    if $d[v] > d[u] + w(u, v)$ **then**
      $d[v] \leftarrow d[u] + w(u, v)$
      $\pi[v] \leftarrow u$

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 weights 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**
      $d[v] \leftarrow d[u] + w(u, v)$
      $\pi[v] \leftarrow u$

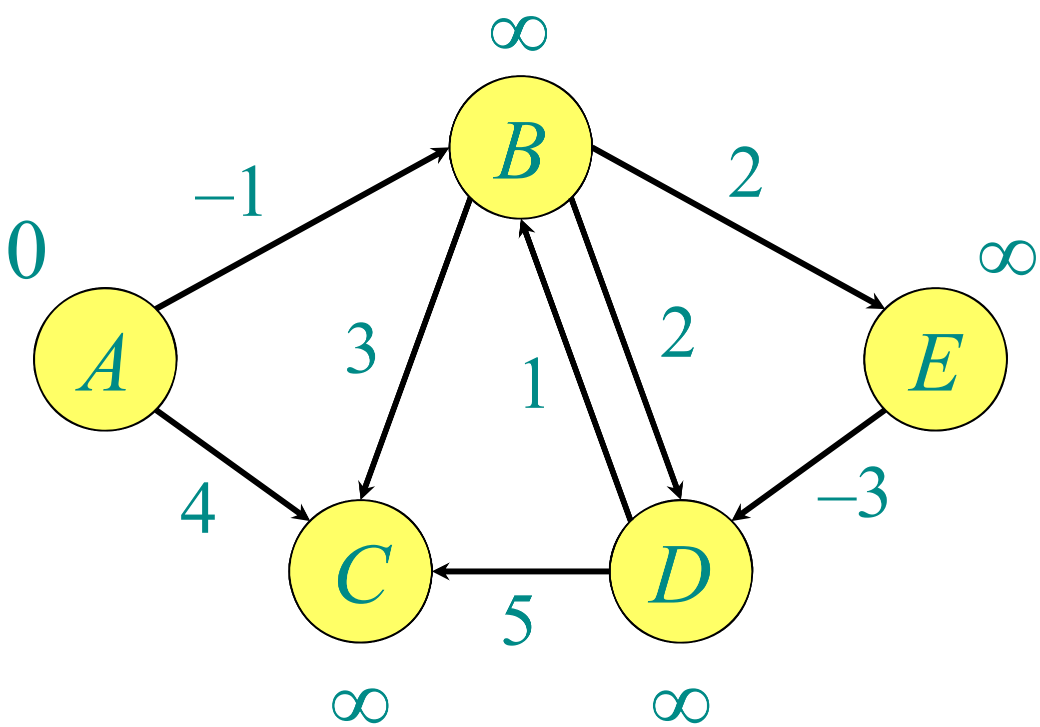
relaxation step

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, $d[v] = \delta(s, v)$. Time $= O(|V||E|)$.

Example of Bellman-Ford

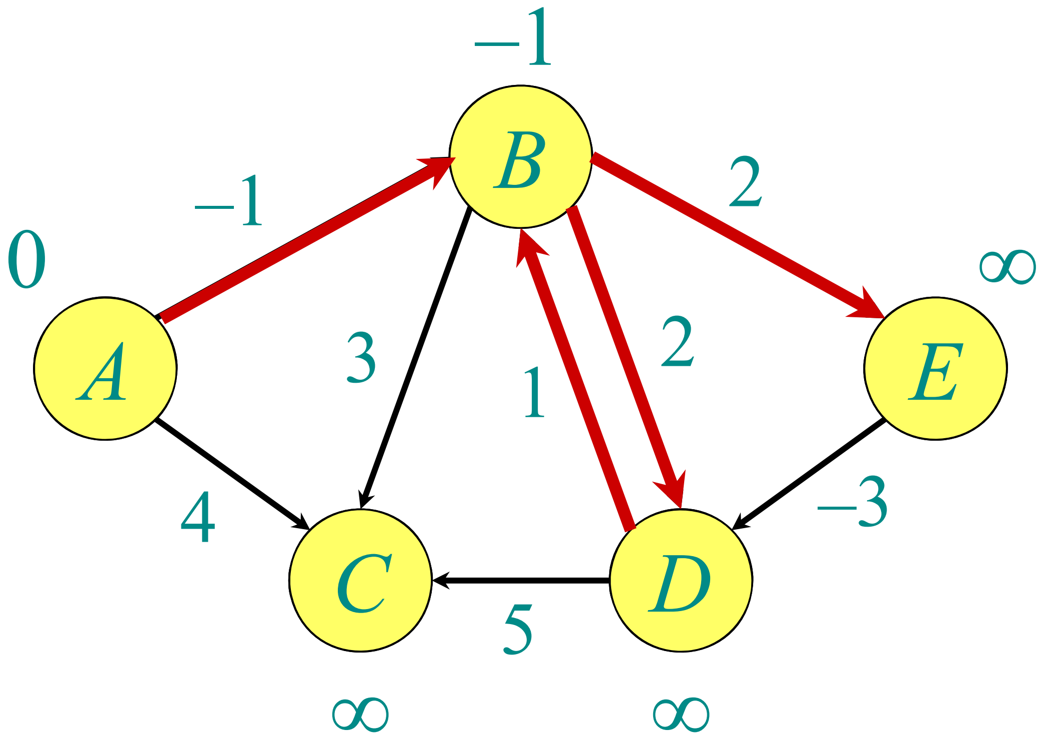
Order of edges: *(B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D)*



A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞

Example of Bellman-Ford

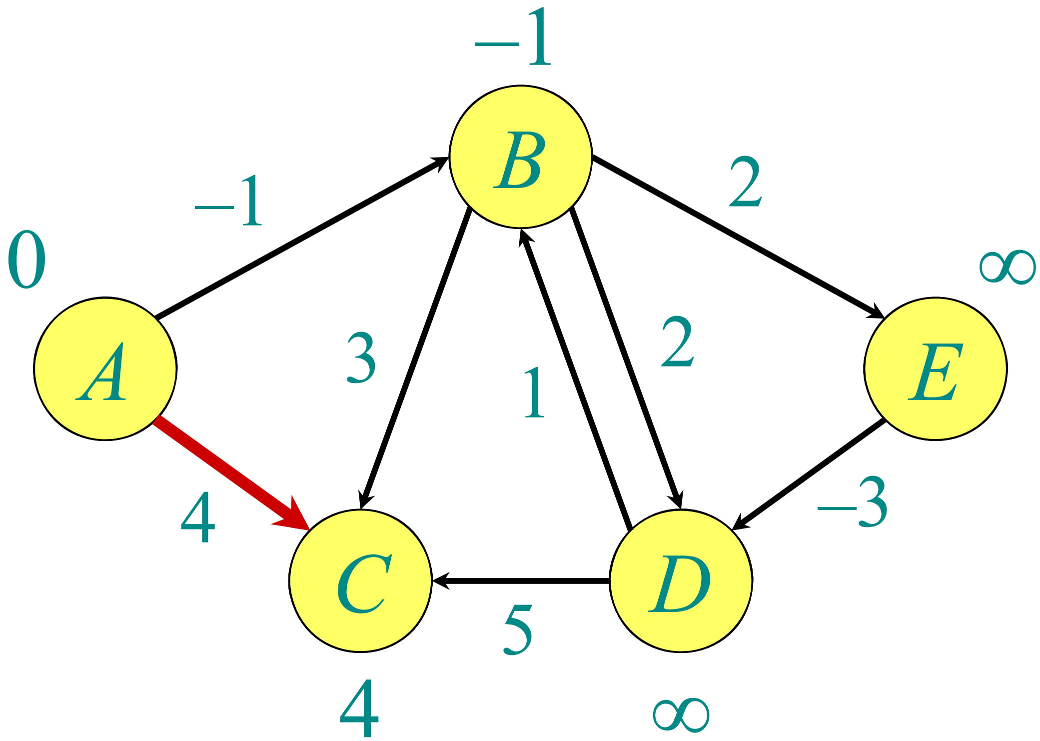
Order of edges: *(B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D)*



A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞
0	–1	∞	∞	∞

Example of Bellman-Ford

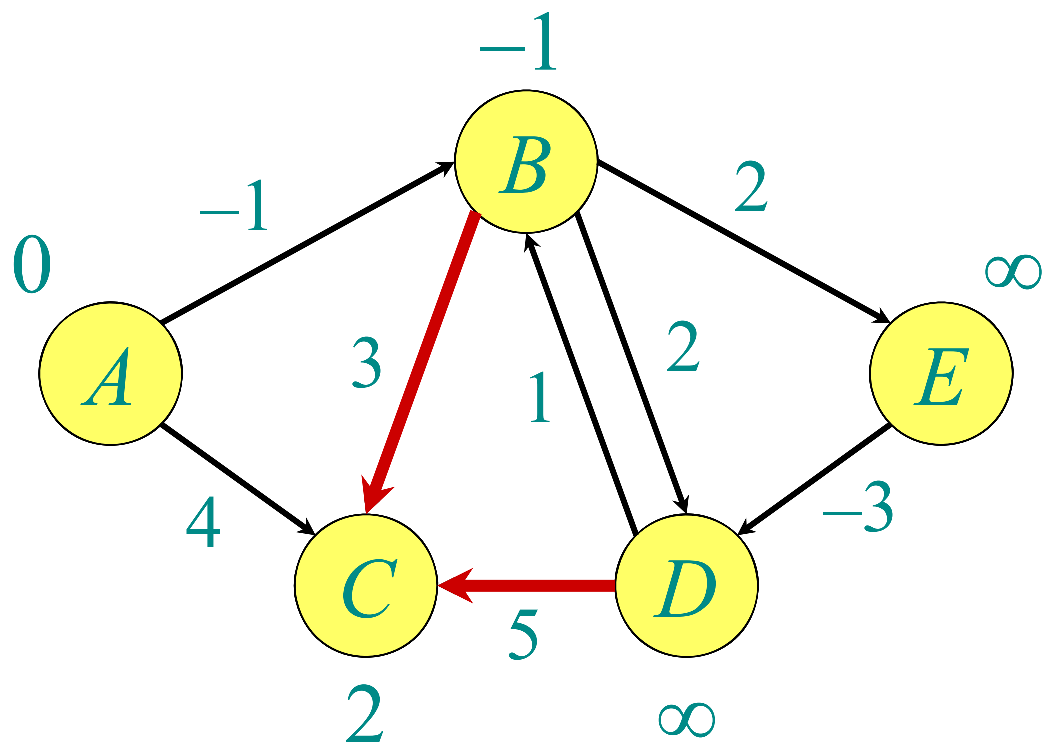
Order of edges: *(B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D)*



A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞
0	–1	∞	∞	∞
0	–1	4	∞	∞

Example of Bellman-Ford

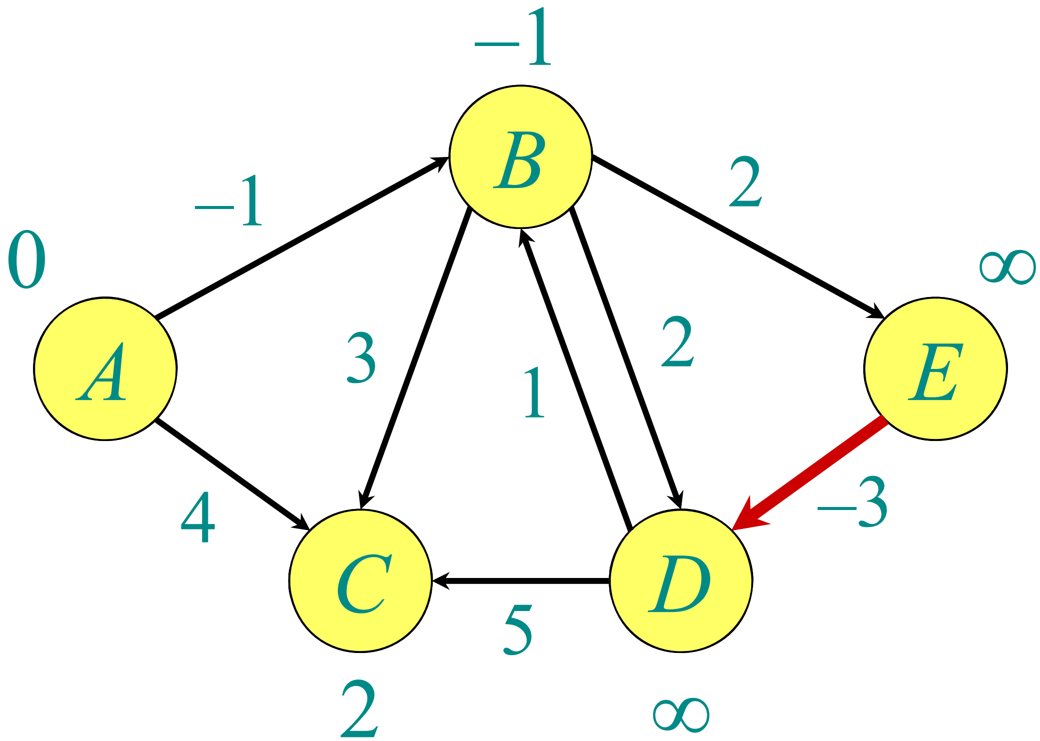
Order of edges: *(B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D)*



A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞
0	–1	∞	∞	∞
0	–1	4	∞	∞
0	–1	2	∞	∞

Example of Bellman-Ford

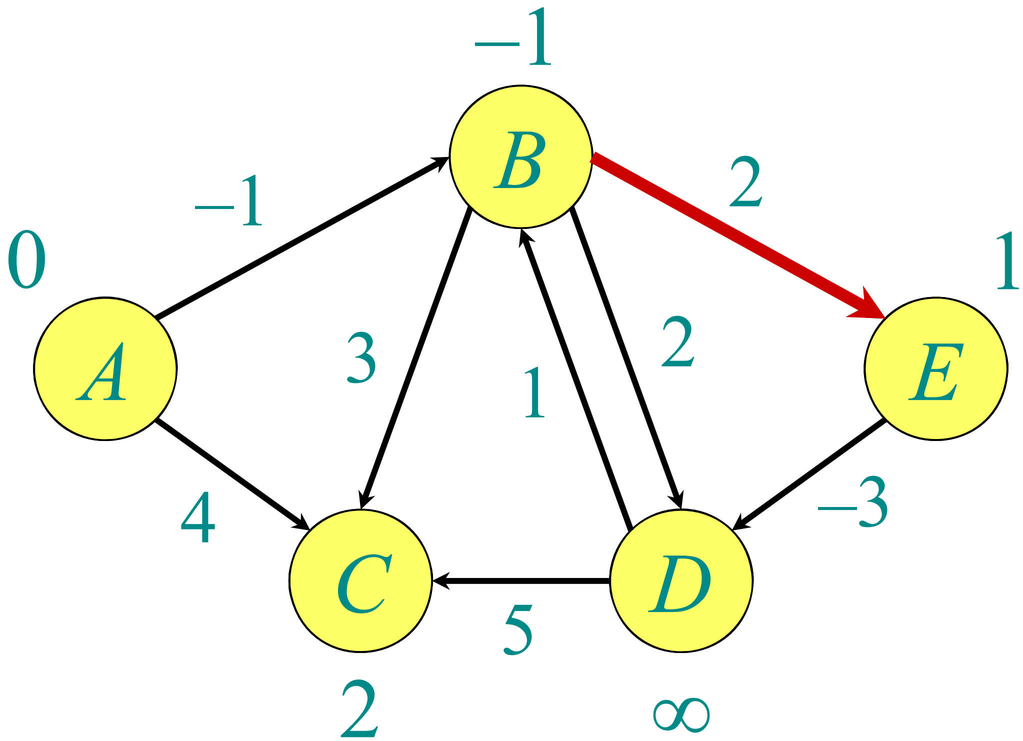
Order of edges: *(B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D)*



A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞
0	–1	∞	∞	∞
0	–1	4	∞	∞
0	–1	2	∞	∞

Example of Bellman-Ford

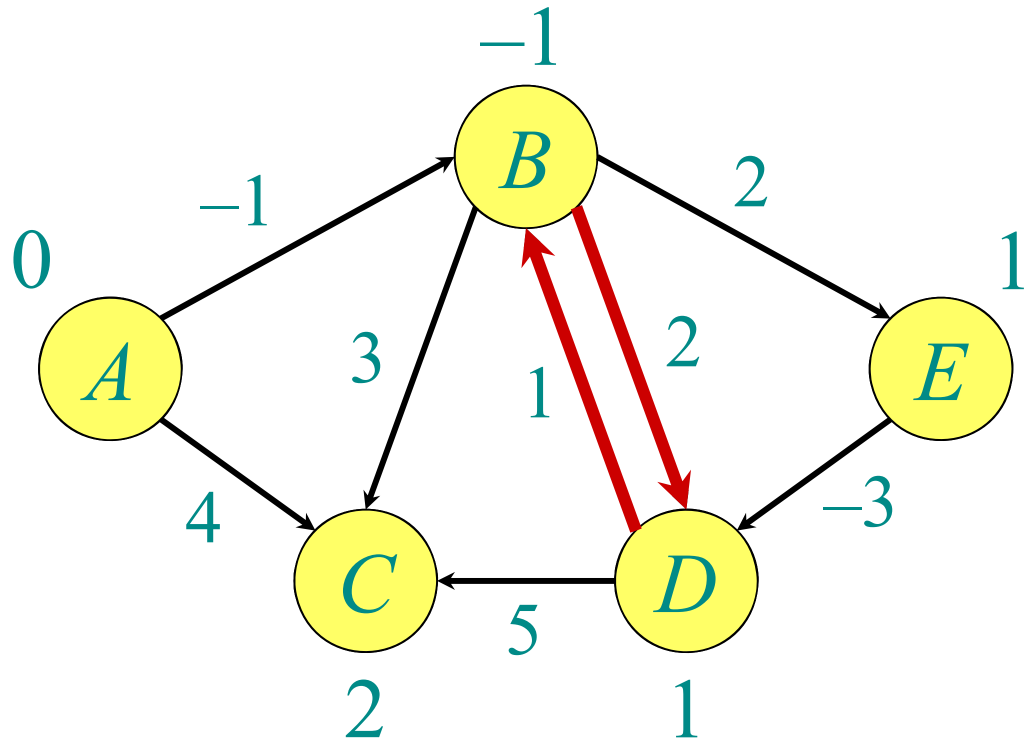
Order of edges: *(B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D)*



A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞
0	–1	∞	∞	∞
0	–1	4	∞	∞
0	–1	2	∞	∞
0	–1	2	∞	1

Example of Bellman-Ford

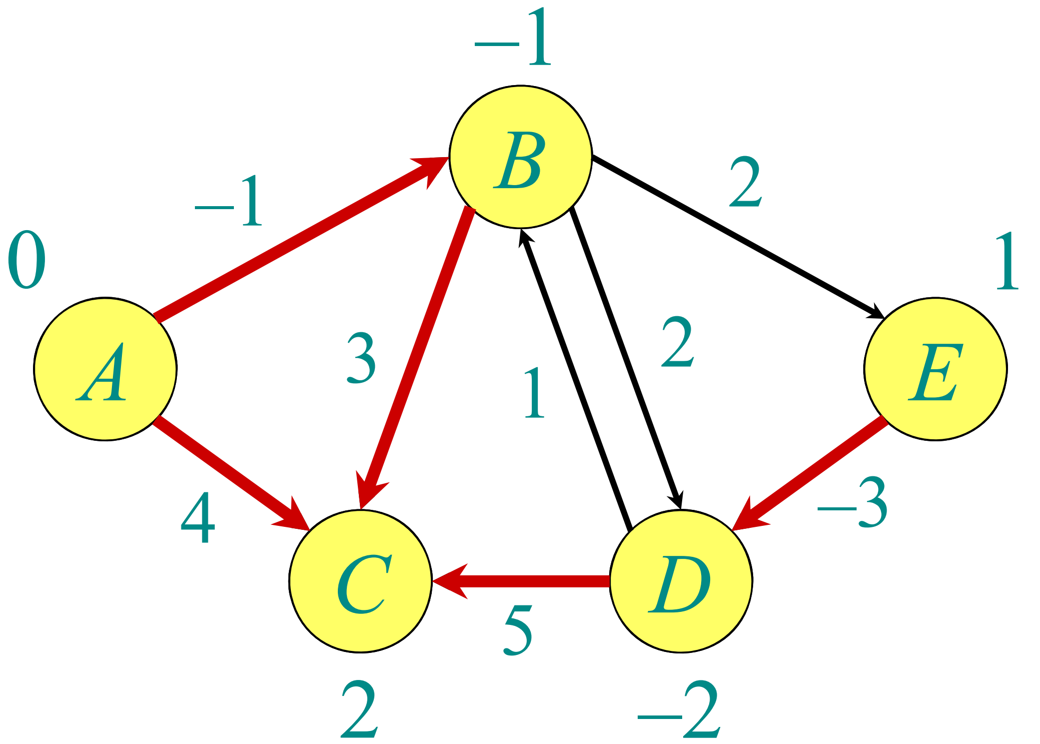
Order of edges: *(B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D)*



A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞
0	–1	∞	∞	∞
0	–1	4	∞	∞
0	–1	2	∞	∞
0	–1	2	∞	1
0	–1	2	1	1

Example of Bellman-Ford

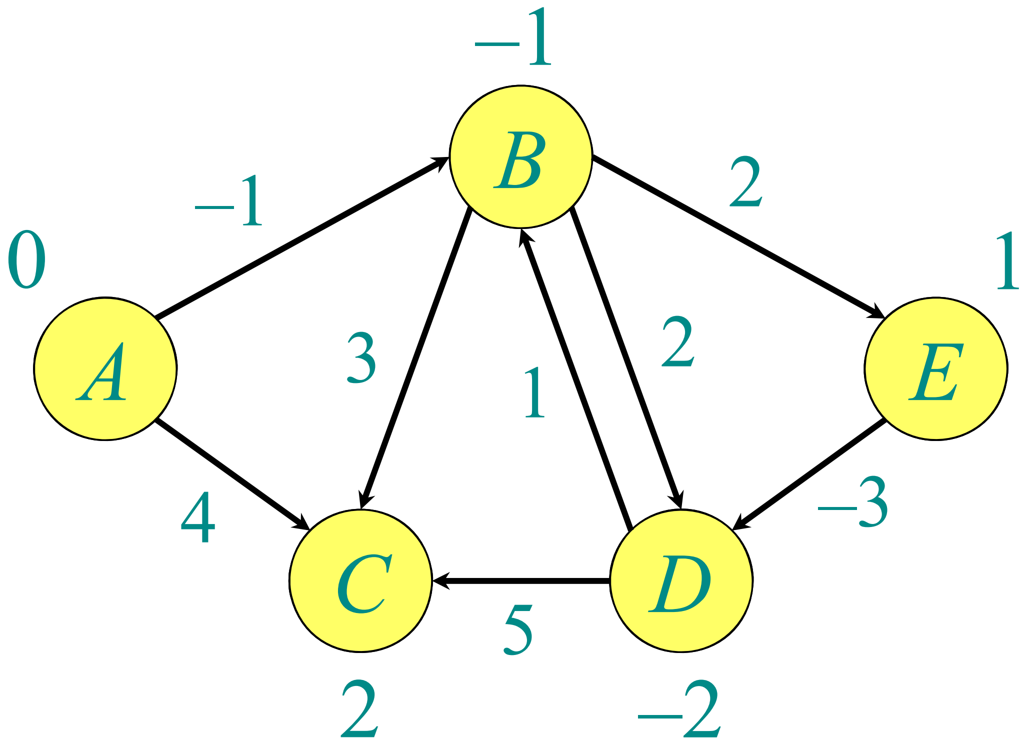
Order of edges: *(B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D)*



A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞
0	–1	∞	∞	∞
0	–1	4	∞	∞
0	–1	2	∞	∞
0	–1	2	∞	1
0	–1	2	1	1
0	–1	2	–2	1

Example of Bellman-Ford

Order of edges: *(B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D)*



A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞
0	–1	∞	∞	∞
0	–1	4	∞	∞
0	–1	2	∞	∞
0	–1	2	∞	1
0	–1	2	1	1
0	–1	2	–2	1

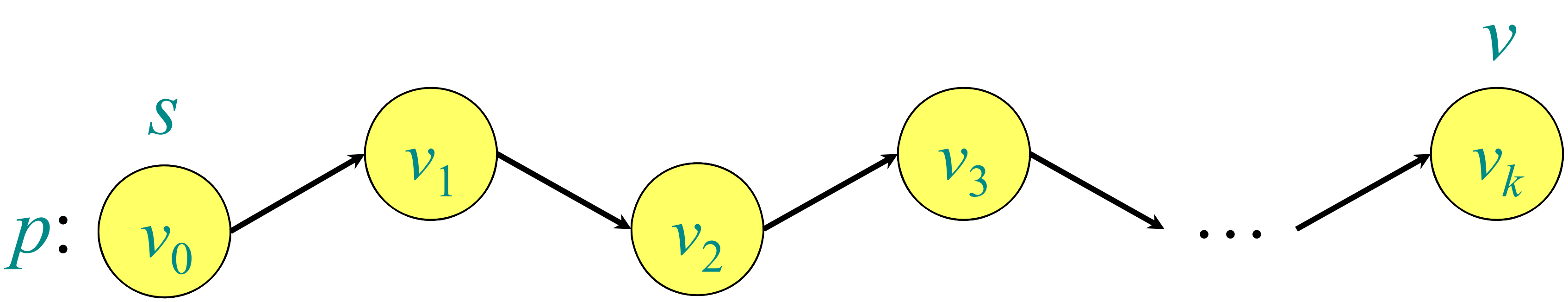
… and 2 more iterations

Note: *d*-values decrease monotonically.

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 $v \in V$.

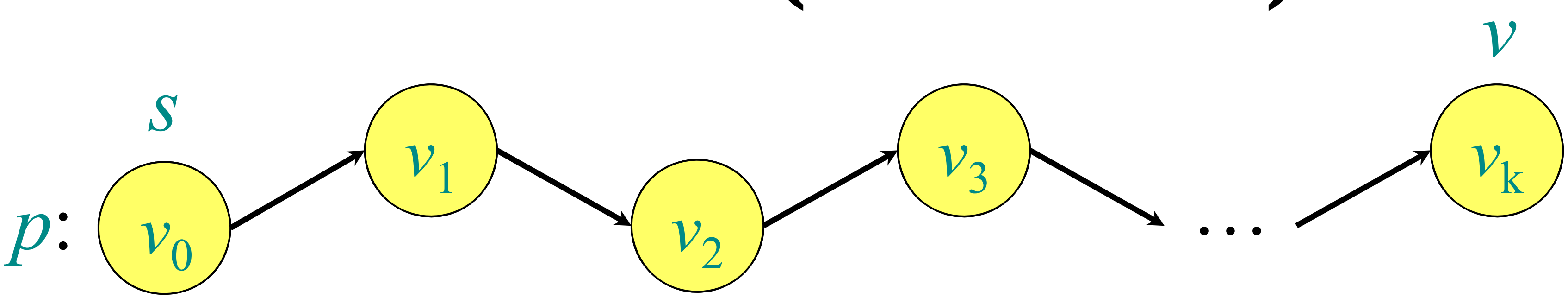
Proof. Let $v \in V$ be any vertex, and consider a shortest path p from s to v with the minimum number of edges.



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 $d[s]$ is unchanged by subsequent relaxations.

- 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. ∎

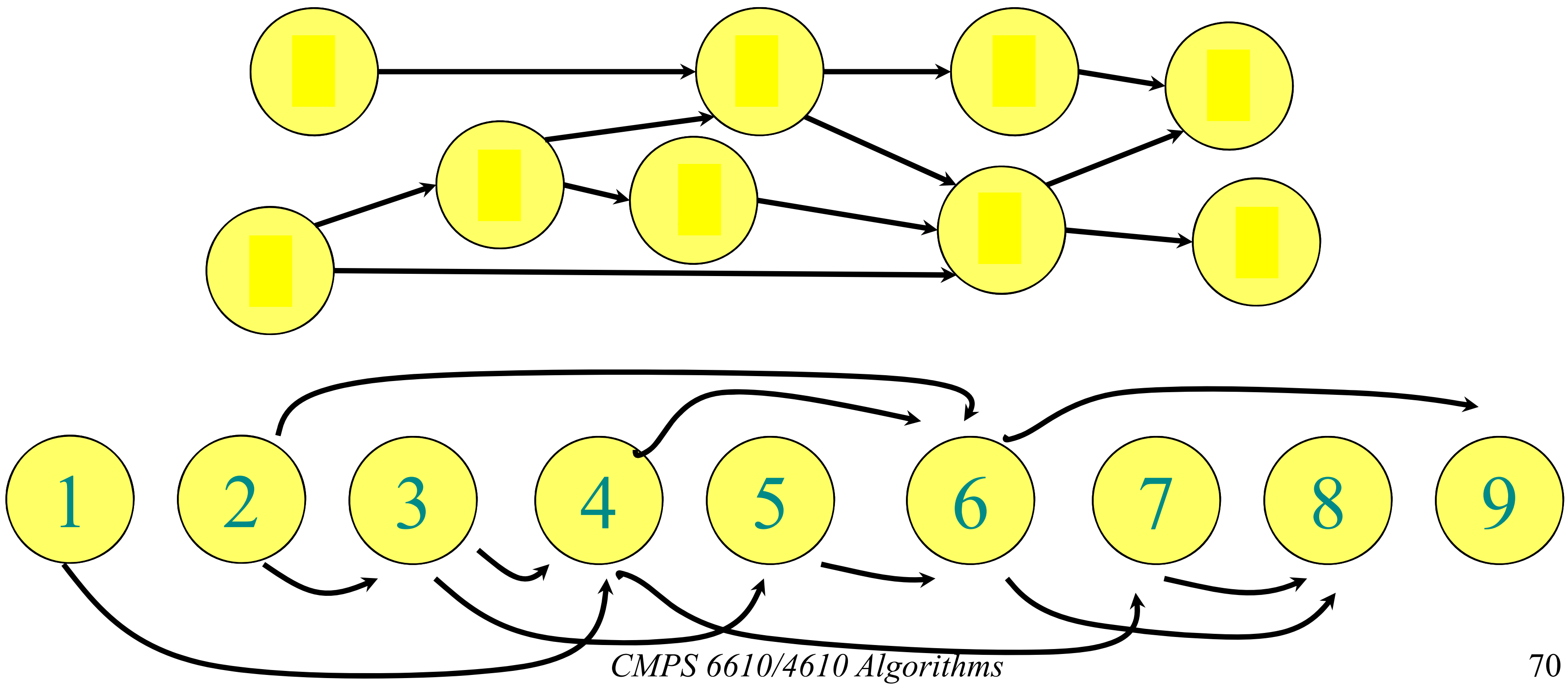
Detection of negative-weight cycles

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

DAG shortest paths

If the graph is a ***directed acyclic graph*** (***DAG***), we first ***topologically sort*** the vertices.

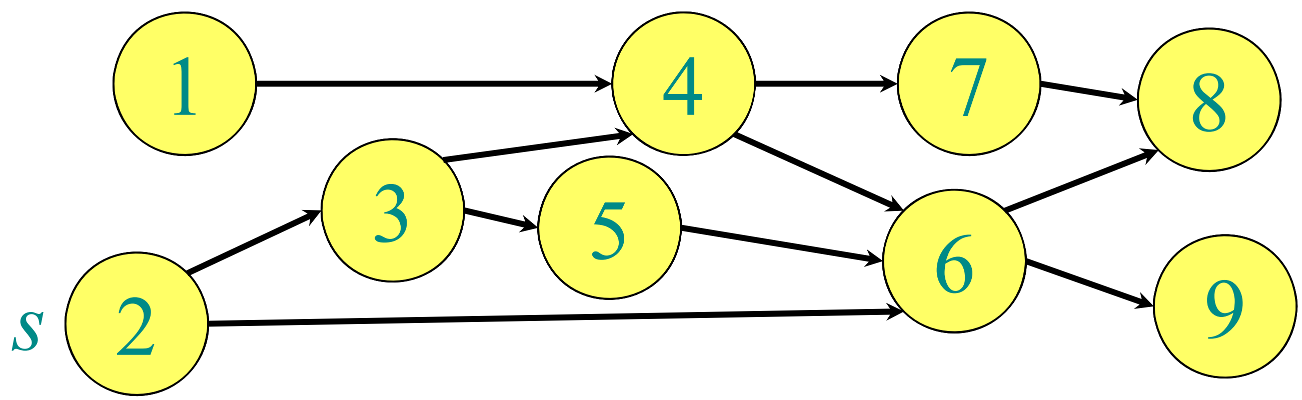
- Determine $f : V \to \{1, 2, \ldots, |V|\}$ such that $(u, v) \in E \Rightarrow f(u) < f(v)$.



DAG shortest paths

If the graph is a ***directed acyclic graph*** (***DAG***), we first ***topologically sort*** the vertices.

- Determine $f : V \rightarrow \{1, 2, \ldots, |V|\}$ such that $(u, v) \in E$ $\Rightarrow f(u) < f(v)$.
- $O(|V| + |E|)$ time



- Walk through the vertices $u \in V$ in this order, relaxing the edges in $Adj[u]$, thereby obtaining the shortest paths from s in a total of $O(|V| + |E|)$ time.

Shortest paths

Single-source shortest paths

- Nonnegative edge weights
    - Dijkstra's algorithm: $O(|E| + |V| \log |V|)$
- General: Bellman-Ford: $O(|V||E|)$
- DAG: One pass of Bellman-Ford: $O(|V| + |E|)$

All-pairs shortest paths

All-pairs shortest paths

Input: Digraph $G = (V, E)$, where $|V| = n$, with edge-weight function $w : E \rightarrow \mathbb{R}$.
Output: $n \times n$ matrix of shortest-path lengths $\delta(i, j)$ for all $i, j \in V$.

Algorithm #1:

- Run Bellman-Ford once from each vertex.
- Time $= O(|V|^2 |E|)$.
- But: Dense graph $\Rightarrow O(|V|^4)$ time.

Shortest paths

Single-source shortest paths

- Nonnegative edge weights
  - Dijkstra's algorithm: $O(|E| + |V| \log |V|)$
- General: Bellman-Ford: $O(|V||E|)$
- DAG: One pass of Bellman-Ford: $O(|V| + |E|)$

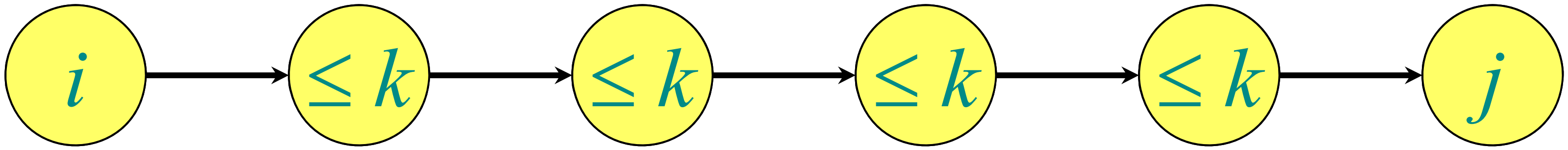
All-pairs shortest paths

- Nonnegative edge weights
  - Dijkstra's algorithm $|V|$ times: $O(|V||E|+|V|^2 \log |V|)$
- General
  - Bellman-Ford $|V|$ times: $\mathrm{O}(|V|^2 |E|)$
  - Floyd-Warshall: $\mathrm{O}(|V|^3)$

Floyd-Warshall algorithm

- Dynamic programming algorithm.
- Assume $V=\{1, 2, \ldots, n\}$, and assume G is given in an **adjacency matrix** $A=(a_{ij})_{1\le i,j\le n}$ where a_{ij} is the weight of the edge from i to j.

Define $c_{ij}^{(k)}$ = weight of a shortest path from i to j with intermediate vertices belonging to the set $\{1, 2, \ldots, k\}$.



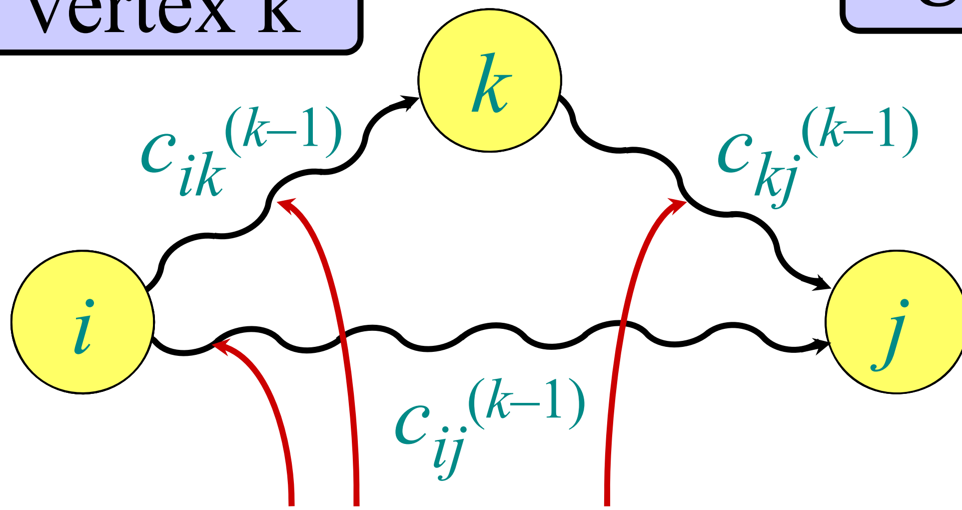
Thus, $\delta(i, j) = c_{ij}^{(n)}$. Also, $c_{ij}^{(0)} = a_{ij}$.

Floyd-Warshall recurrence

$$c_{ij}^{(k)} = \min \{c_{ij}^{(k-1)}, c_{ik}^{(k-1)} + c_{kj}^{(k-1)}\}$$

Do not use vertex k

Use vertex k



intermediate vertices in $\{1, 2, \ldots, k\text{-}1\}$

Pseudocode for Floyd-Warshall

for $k \leftarrow 1$ **to** n **do**
  for $i \leftarrow 1$ **to** n **do**
    for $j \leftarrow 1$ **to** n **do**
      if $c_{ij}^{(k-1)} > c_{ik}^{(k-1)} + c_{kj}^{(k-1)}$ **then**
        $c_{ij}^{(k)} \leftarrow c_{ik}^{(k-1)} + c_{kj}^{(k-1)}$
      else
        $c_{ij}^{(k)} \leftarrow c_{ij}^{(k-1)}$

relaxation (the if/then step)

- Runs in $\Theta(n^3)$ time and space
- Simple to code.
- Efficient in practice.

Transitive Closure of a Directed Graph

Compute $t_{ij} = \begin{cases} 1 & \text{if there exists a path from } i \text{ to } j, \\ 0 & \text{otherwise.} \end{cases}$

IDEA: Use Floyd-Warshall, but with $(\vee, \wedge)$ instead of $(\min, +)$:

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}).$$

Time $= \Theta(n^3)$.

Floyd-Warshall recurrence

$$c_{ij}^{(k)} = \min \{c_{ij}^{(k-1)}, c_{ik}^{(k-1)} + c_{kj}^{(k-1)}\}$$

Shortest paths

Single-source shortest paths

- Nonnegative edge weights
  - Dijkstra's algorithm: $O(|E| + |V| \log |V|)$
- General: Bellman-Ford: $O(|V||E|)$
- DAG: One pass of Bellman-Ford: $O(|V| + |E|)$

} adj. list

All-pairs shortest paths

- Nonnegative edge weights — adj. list
  - Dijkstra's algorithm $|V|$ times: $O(|V||E|+|V|^2 \log |V|)$
- General
  - Bellman-Ford $|V|$ times: $O(|V|^2 |E|)$ — adj. list
  - Floyd-Warshall: $O(|V|^3)$ — adj. matrix

Graph reweighting

Theorem. Given a label $h(v)$ for each $v \in V$, ***reweight*** each edge $(u, v) \in E$ by

$$\hat{w}(u, v) = w(u, v) + h(u) - h(v).$$

Then, all paths between the same two vertices are reweighted by the same amount.

Proof. Let $p = v_1 \to v_2 \to \cdots \to v_k$ be a path in the graph.

Then, we have

$$\begin{aligned}
\hat{w}(p) &= \sum_{i=1}^{k-1} \hat{w}(v_i, v_{i+1}) \\
&= \sum_{i=1}^{k-1} \big(w(v_i, v_{i+1}) + h(v_i) - h(v_{i+1}) \big) \\
&= \sum_{i=1}^{k-1} w(v_i, v_{i+1}) + h(v_1) - h(v_k) \\
&= w(p) + h(v_1) - h(v_k).
\end{aligned}$$

□

Johnson's algorithm

1. Find a vertex labeling h, by running Bellman-Ford on G + super-source s. Set $h(v)=\delta(s,v)$ or determine that a negative-weight cycle exists.
   By triangle inequality $h(v) \le h(u) + w(u, v)$, and hence $\hat{w}(u, v) = w(u, v) + h(u) - h(v) \ge 0$.
   - Time $= O(|V||E|)$
2. Run Dijkstra's algorithm from each vertex using $\hat{w}$.
   - Time $= O(|V||E|+|V|^2 \log |V|)$.
3. Reweight each shortest-path weight $\hat{\delta}(u,v)$ to compute the shortest-path weight $\delta(u,v) = \hat{\delta}(u,v) - h(u) + h(v)$ of the original graph G.
   - Time $= O(|V|^2)$

Total time $= O(|V||E|+|V|^2 \log |V|)$.

Shortest paths

Single-source shortest paths

- Nonnegative edge weights
  - Dijkstra's algorithm: $O(|E| + |V| \log |V|)$
- General: Bellman-Ford: $O(|V||E|)$
- DAG: One pass of Bellman-Ford: $O(|V| + |E|)$

} adj. list

All-pairs shortest paths

- Nonnegative edge weights
  - Dijkstra's algorithm $|V|$ times: $O(|V||E|+|V|^2 \log |V|)$ — adj. list
- General
  - Bellman-Ford $|V|$ times: $O(|V|^2 |E|)$ — adj. list
  - Floyd-Warshall: $O(|V|^3)$ — adj. matrix
  - Johnson's algorithm: $O(|V||E|+|V|^2 \log |V|)$ — adj. list