香港科技大学(广州)
THE HONG KONG
UNIVERSITY OF SCIENCE AND
TECHNOLOGY (GUANGZHOU)

Design and Analysis of Algorithms

Jing Tang | DSAA 2043 Fall 2024

Shortest Paths

Objective

- Define the single source shortest path problem.
- Optimal substructure, Triangle inequality and Relaxation.
- The Dijkstra algorithm and its extension.

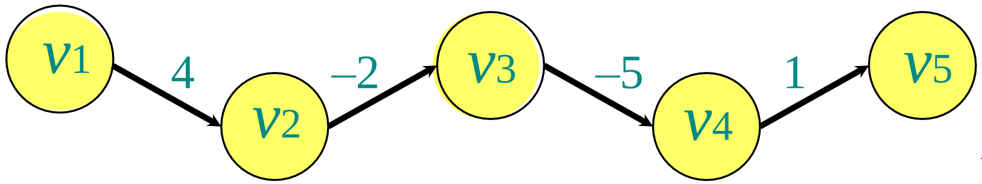
Paths in graphs.

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

The ***weight*** of path $p = v_1 \to v_2 \to \dots \to v_k$ is defined to be

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

Example:



$w(p) = -2$

Paths in graphs.

A ***shortest path*** from *u* to *v* is a path of minimum weight from *u* to *v*.

The ***shortest-path weight*** from *u* to *v* is defined as:

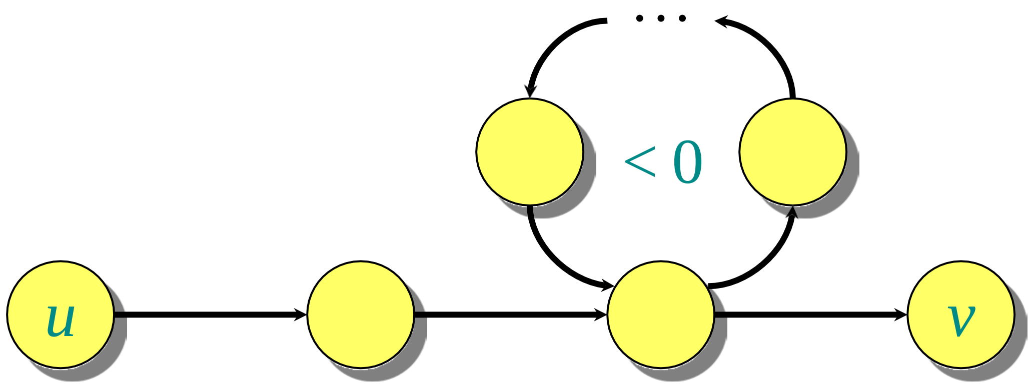
$$\delta(u, v) = \min\{w(p) : p \text{ is a path from } u \text{ to } v\}.$$

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

Well-definedness of shortest paths

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

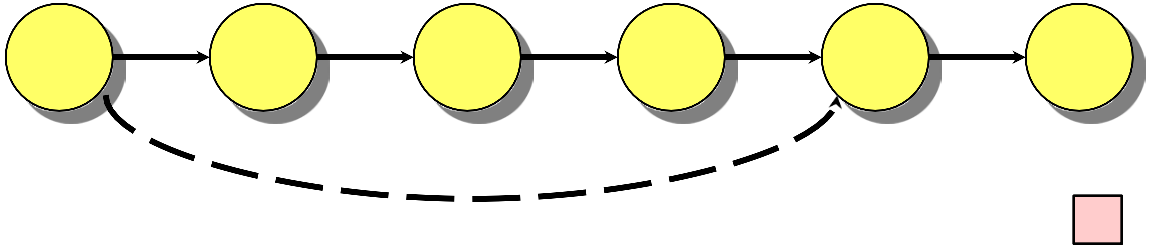
Example:



Optimal substructure

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

Proof. Cut and paste:



If v_j on optimal path from v_0 to v_n: $\delta(v_0, v_n) = \delta(v_0, v_j) + \delta(v_j, v_n)$.

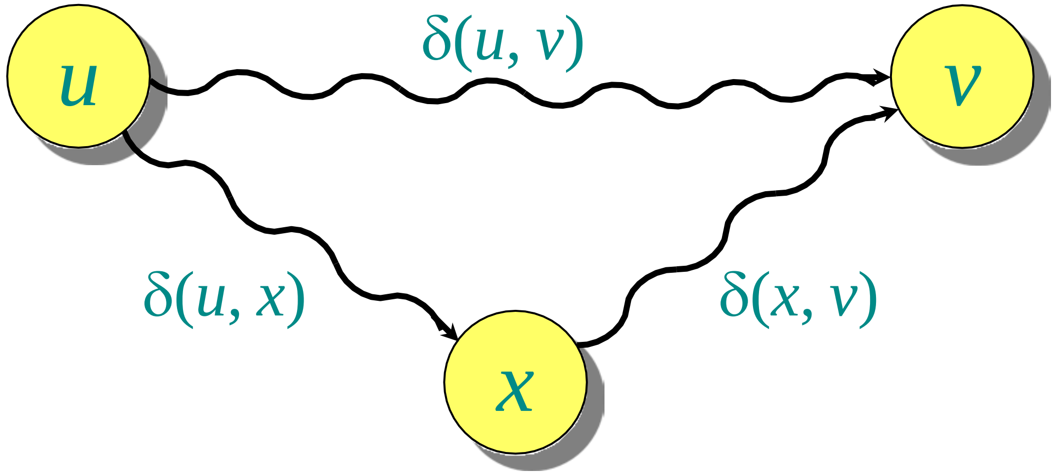
If the sub-path v_i to v_j is not optimal, then by finding a shorter path from v_i to v_j we can strictly improve the original path.

Triangle inequality

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

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

Proof.



If u not on shortest path from s to t: $\delta(s,t) < \delta(s,u) + \delta(u,t)$.

u is on shortest path from s to t iff $\delta(s,t) = \delta(s,u) + \delta(u,t)$.

Single-source shortest paths

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

IDEA: Greedy.

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

Dijkstra's algorithm

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

Dijkstra's algorithm

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
    do $d[v] \leftarrow \infty$
$S \leftarrow \emptyset$
$Q \leftarrow V$ ⊳ Q is a priority queue maintaining $V - S$, keyed on $d[v]$
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 $d[v] \leftarrow d[u] + w(u, v)$

Dijkstra's algorithm

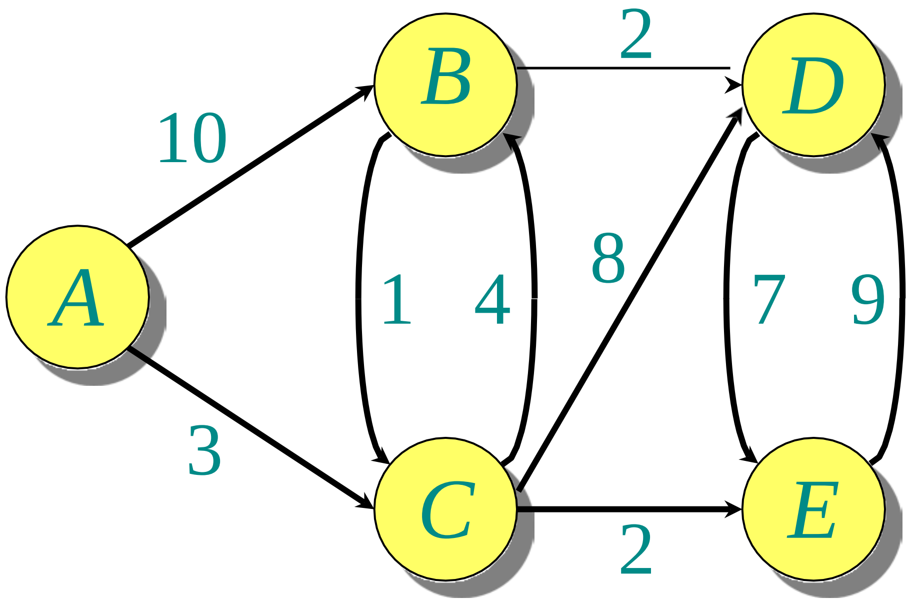
$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
  do $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$ ⊳ Q is a priority queue maintaining $V - S$, keyed on $d[v]$
while $Q \neq \varnothing$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$
    for each $v \in Adj[u]$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

relaxation step

Implicit DECREASE-KEY

Example of Dijkstra's algorithm

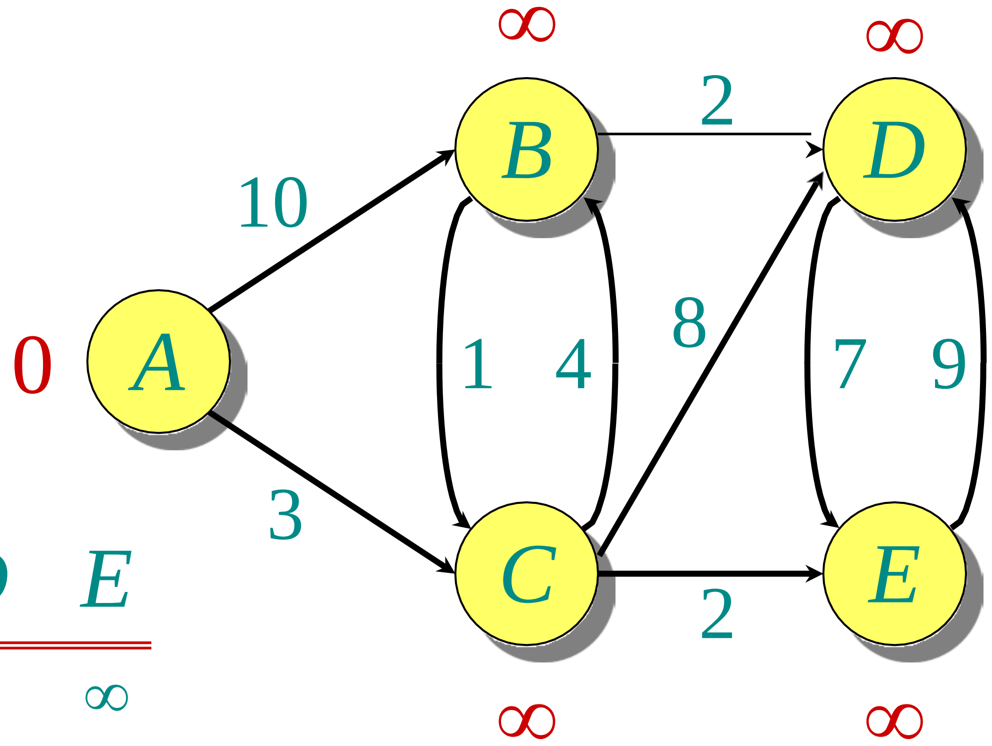
Dijkstra can only handle graphs with nonnegative edge weights:



Try to think why?

Example of Dijkstra's algorithm

Initialize:

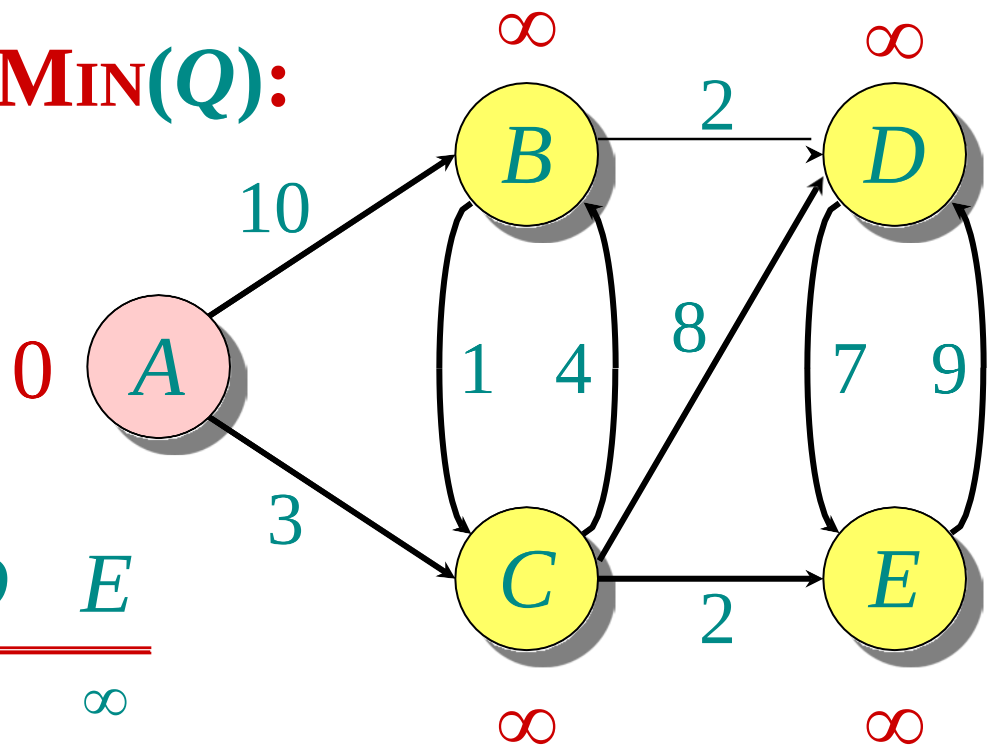


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: {}

Example of Dijkstra's algorithm

"A" ← EXTRACT-MIN(Q):

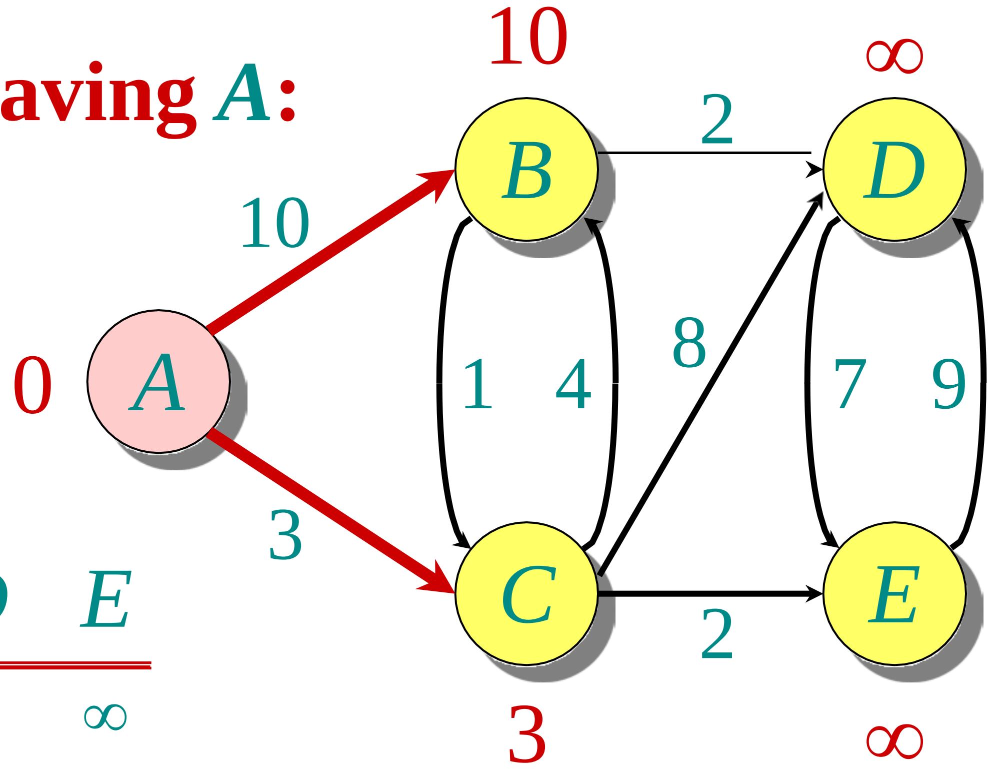


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: { A }

Example of Dijkstra’s algorithm

Relax all edges leaving *A*:

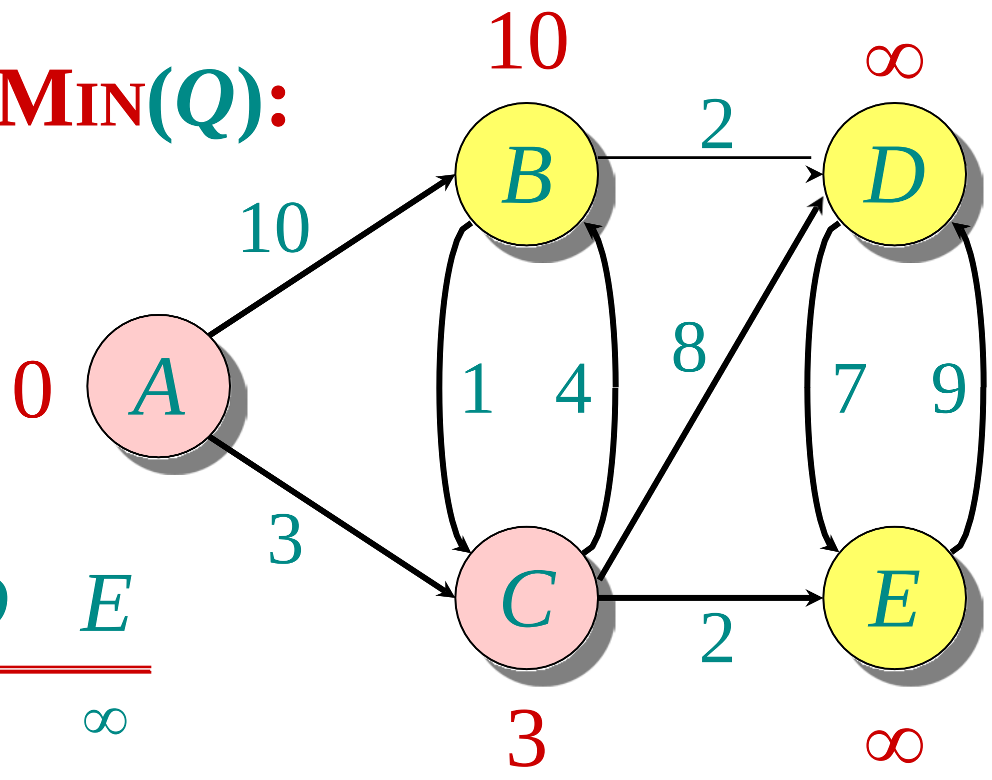


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { *A* }

Example of Dijkstra's algorithm

"C" ← EXTRACT-MIN(Q):



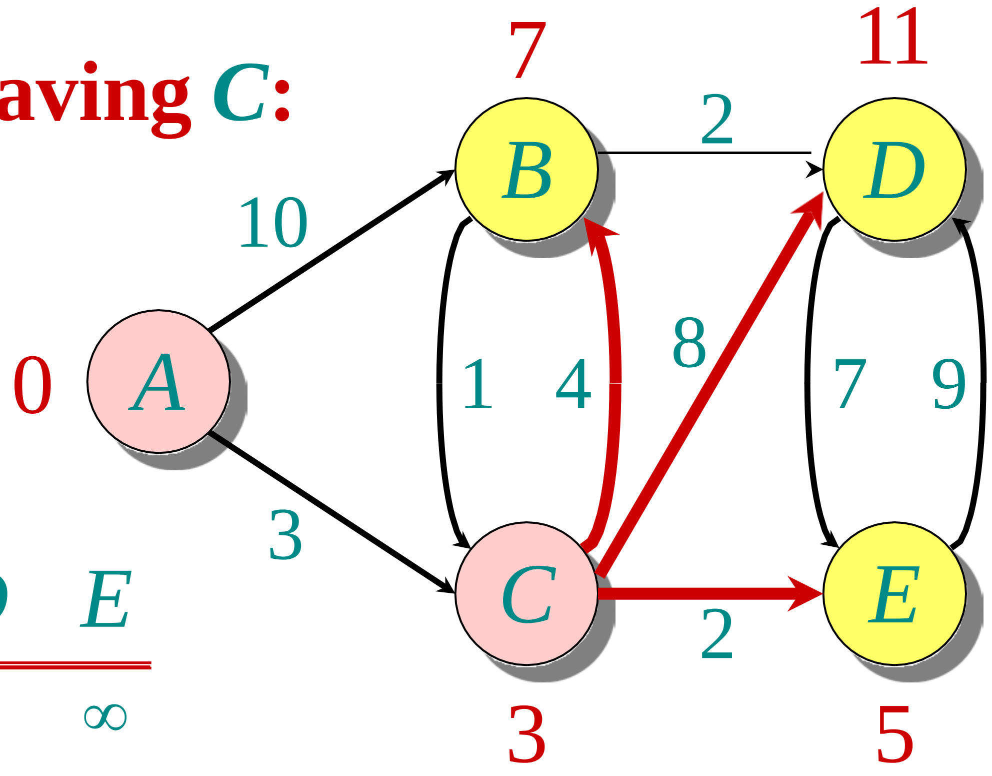
Q: A B C D E

A	B	C	D	E
0	∞	∞	∞	∞
	10	3	∞	∞

S: { A, C }

Example of Dijkstra's algorithm

Relax all edges leaving *C*:

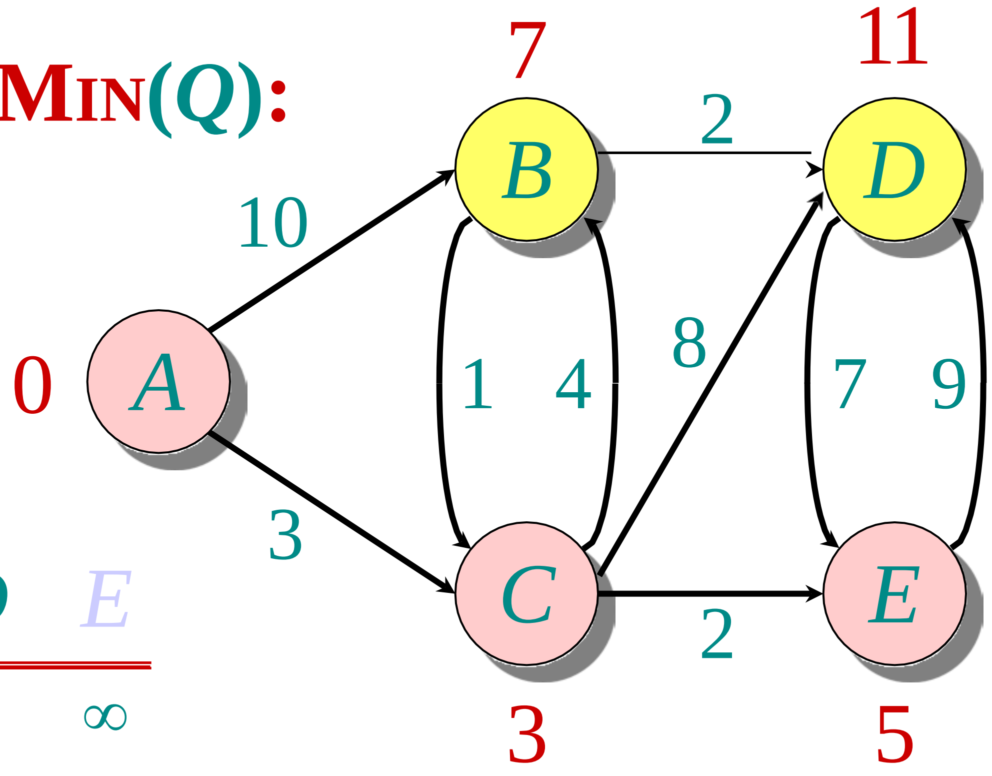


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { *A*, *C* }

Example of Dijkstra’s algorithm

“E” ← EXTRACT-MIN(Q):

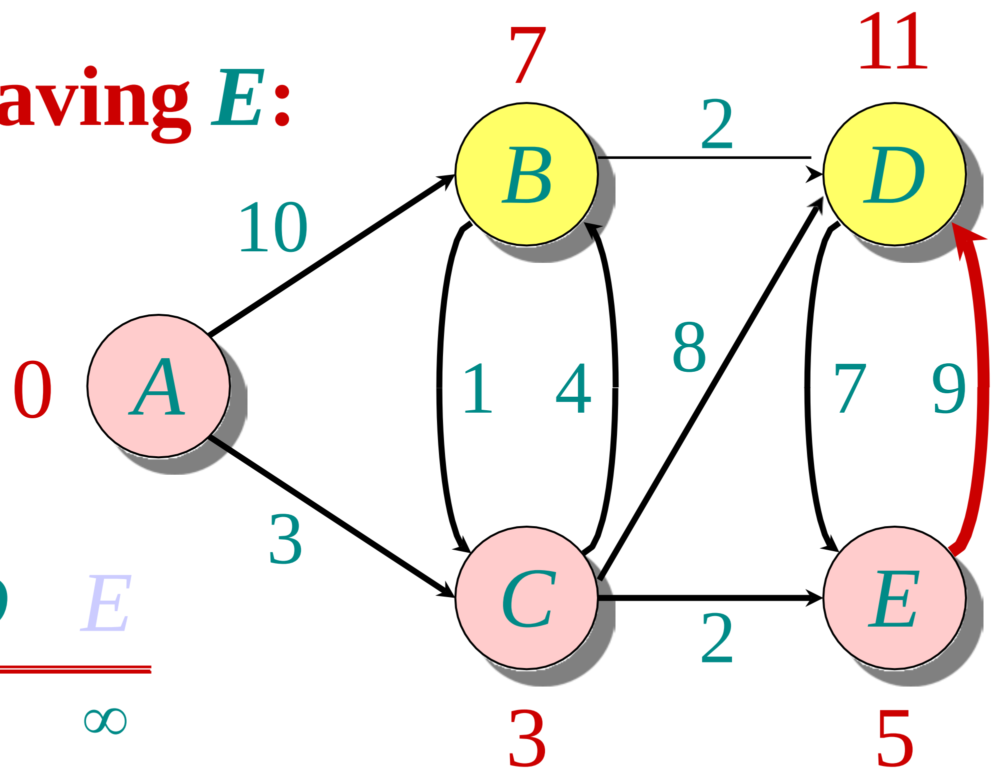


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C, E }

Example of Dijkstra's algorithm

Relax all edges leaving *E*:

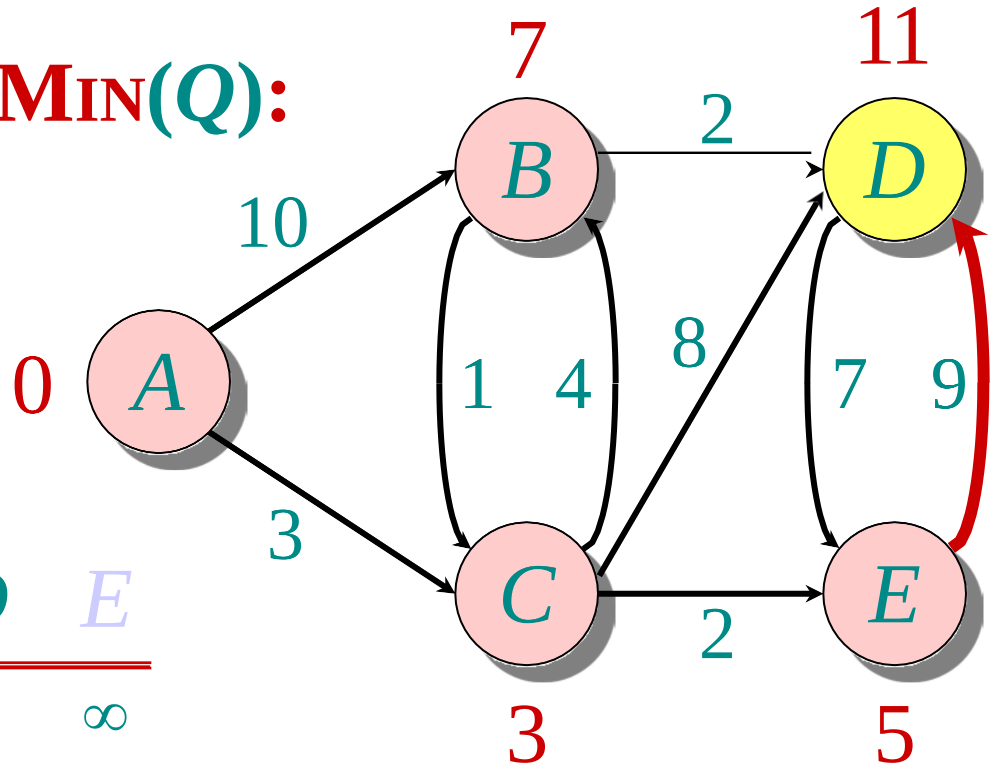


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

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

Example of Dijkstra's algorithm

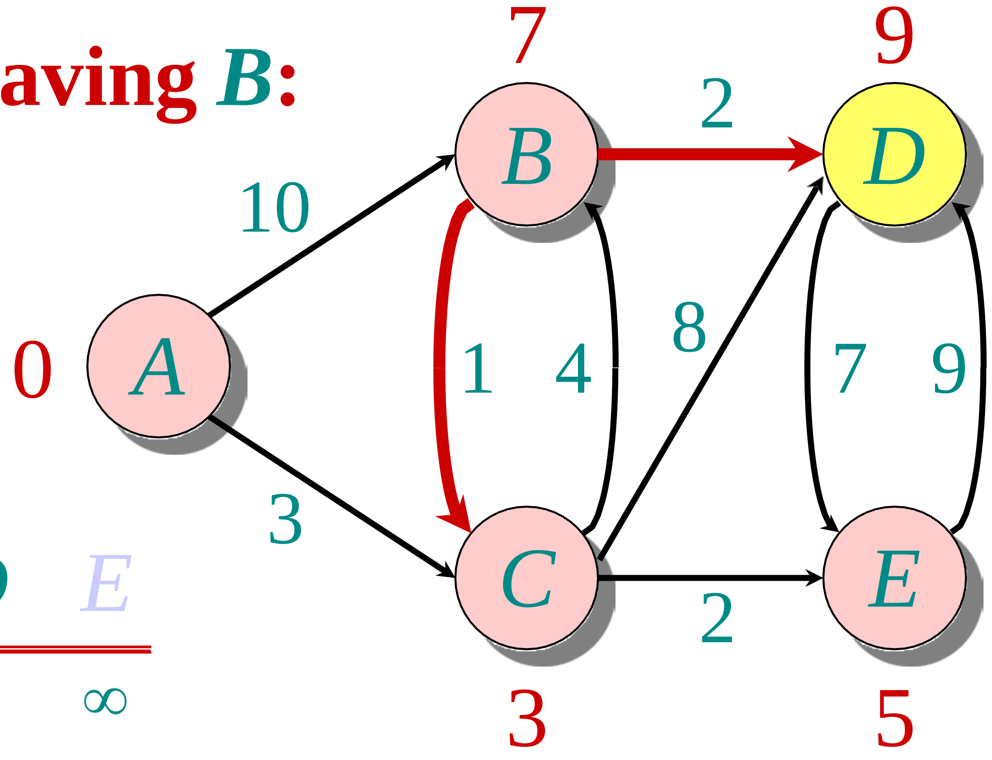
"B" $\leftarrow$ EXTRACT-MIN(Q):



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

S: { A, C, E, B }

Relax all edges leaving *B*:

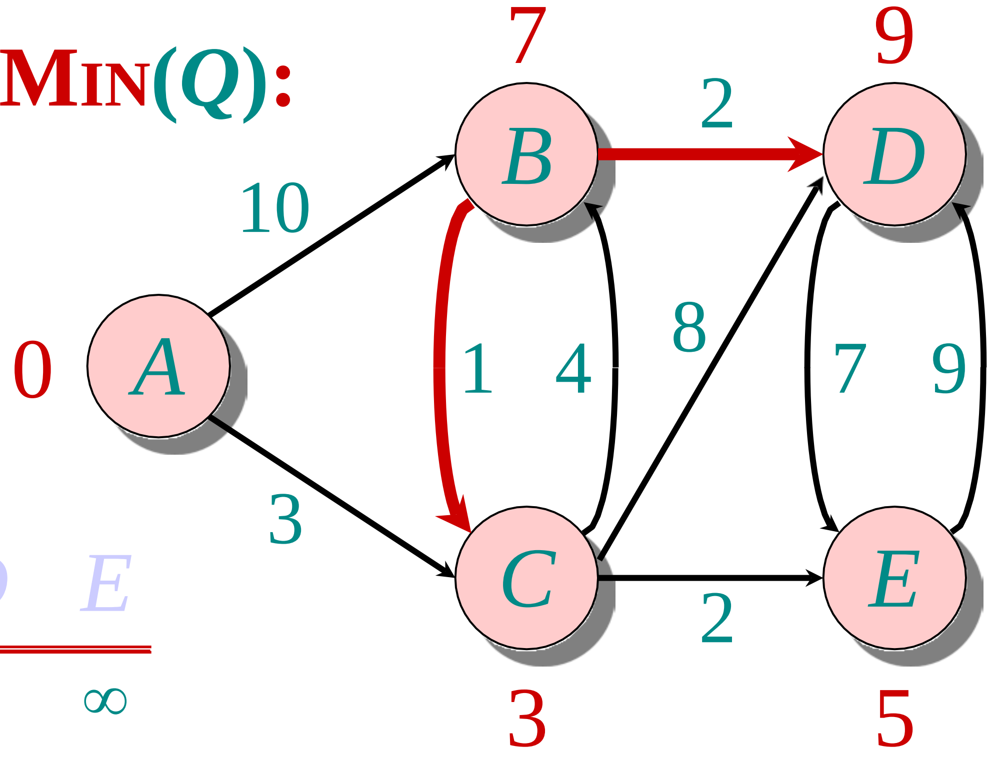


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { A, C, E, B }

Example of Dijkstra's algorithm

"D" ← EXTRACT-MIN(Q):



Q:	A	B	C	D	E
	$\boxed{0}$	∞	∞	∞	∞
		10	$\boxed{3}$	∞	∞
		7		11	$\boxed{5}$
		$\boxed{7}$		11	
				$\boxed{9}$	

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.

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,

$d[v] < \delta(s,v)$	supposition
$\leq \delta(s, u) + \delta(u, v)$	triangle inequality
$\leq \delta(s,u) + w(u, v)$	sh. path $\leq$ specific path
$\leq d[u] + w(u, v)$	v is first violation

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.

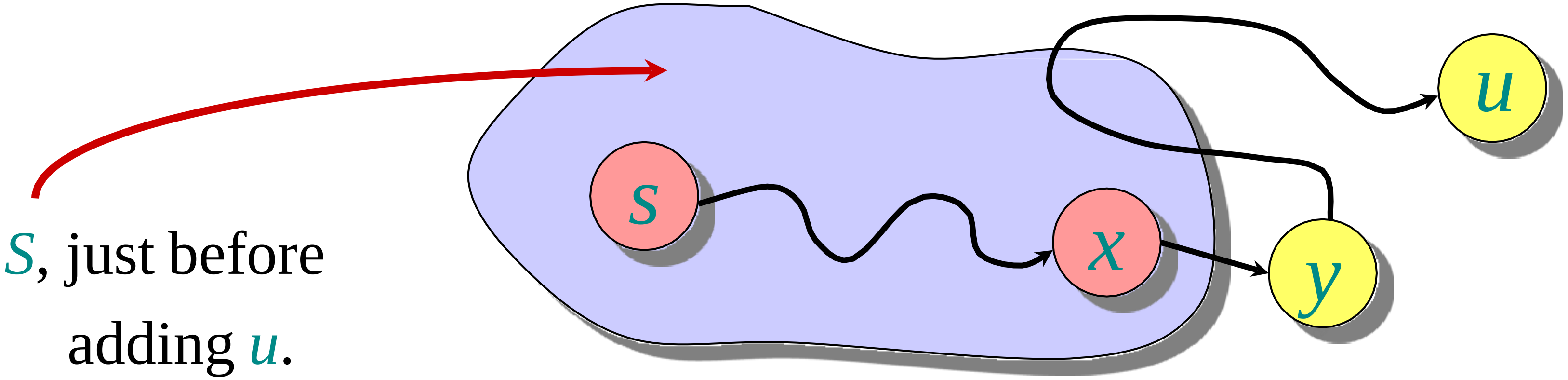
Proof.

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 $d[v] = d[u] + w(u, v) = \delta(s, v)$. $\square$

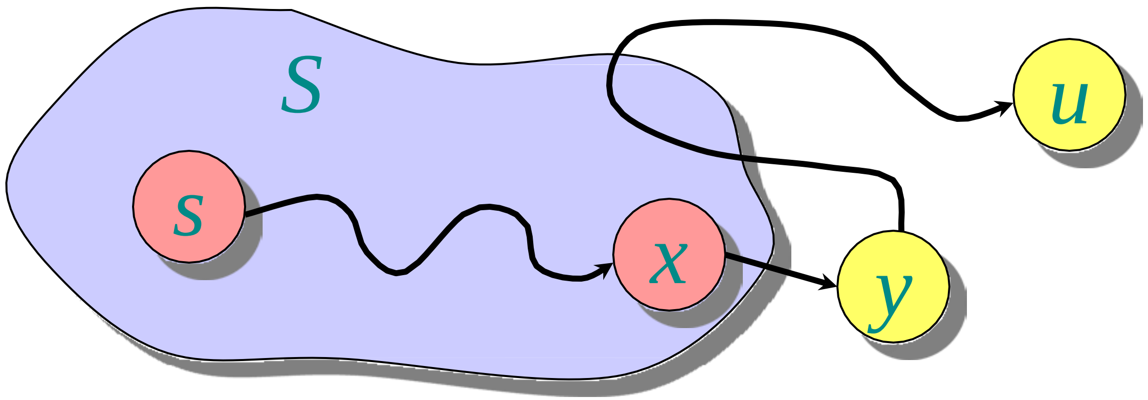
Correctness — Part III

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. $\square$

Analysis of Dijkstra

$$
\left.\begin{array}{l}
\textbf{while } Q \neq \emptyset \\
\quad \textbf{do } u \leftarrow \text{EXTRACT-MIN}(Q) \\
\quad\quad S \leftarrow S \cup \{u\} \\
\quad\quad \textbf{for } \text{each } v \in Adj[u] \\
\quad\quad\quad \textbf{do if } d[v] > d[u] + w(u, v) \\
\quad\quad\quad\quad \textbf{then } d[v] \leftarrow d[u] + w(u, v)
\end{array}\right.
$$

$|V|$ times

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

$degree(u)$ 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.

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

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

Analysis of Dijkstra (continued)

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

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

Analysis of Dijkstra (continued)

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

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

Analysis of Dijkstra (continued)

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

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

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?

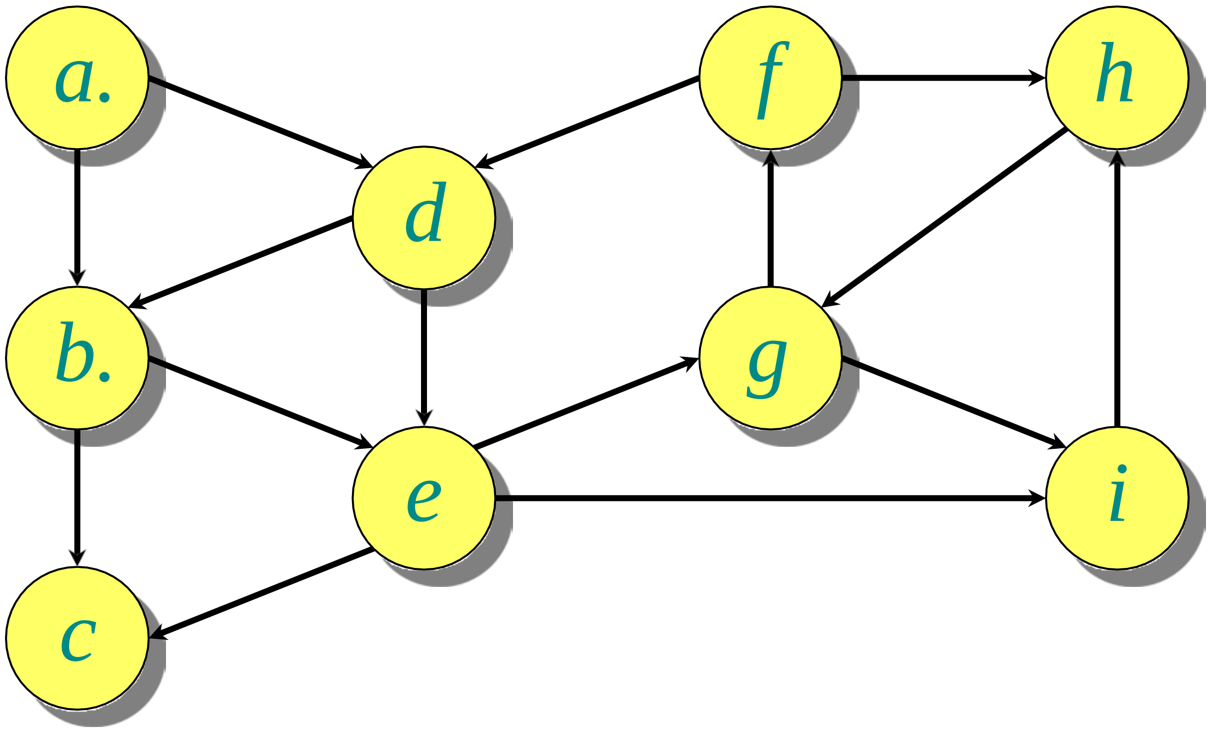
- Use a simple FIFO queue instead of a priority queue.

Breadth-first search

$$
\begin{array}{l}
\textbf{while } Q \neq \emptyset \\
\quad \textbf{do } u \leftarrow \text{DEQUEUE}(Q) \\
\qquad \textbf{for } \text{each } v \in Adj[u] \\
\qquad\quad \textbf{do if } d[v] = \infty \\
\qquad\qquad \textbf{then } d[v] \leftarrow d[u] + 1 \\
\qquad\qquad\quad \text{ENQUEUE}(Q, v)
\end{array}
$$

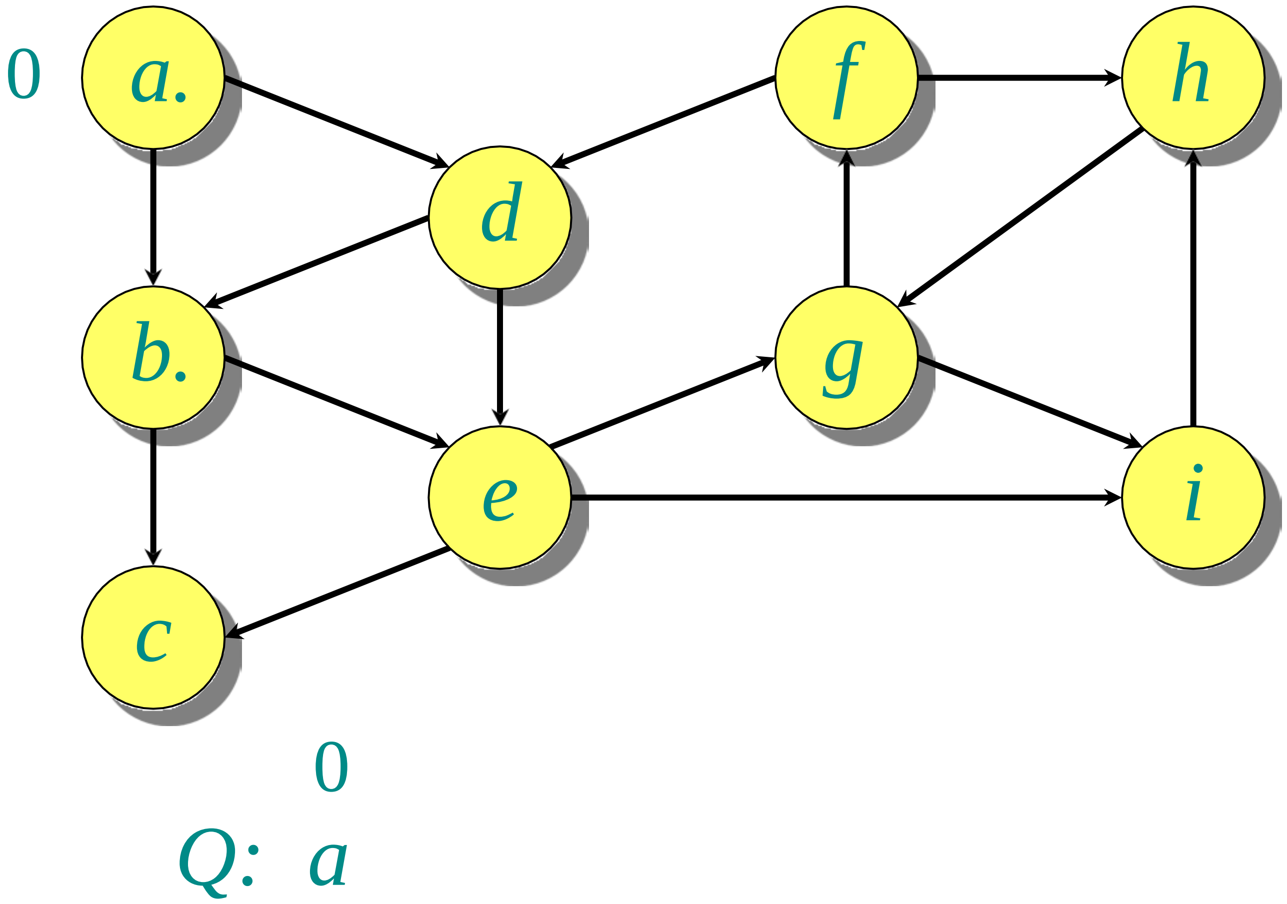
Analysis: Time $= O(|V| + |E|)$.

Example of breadth-first search

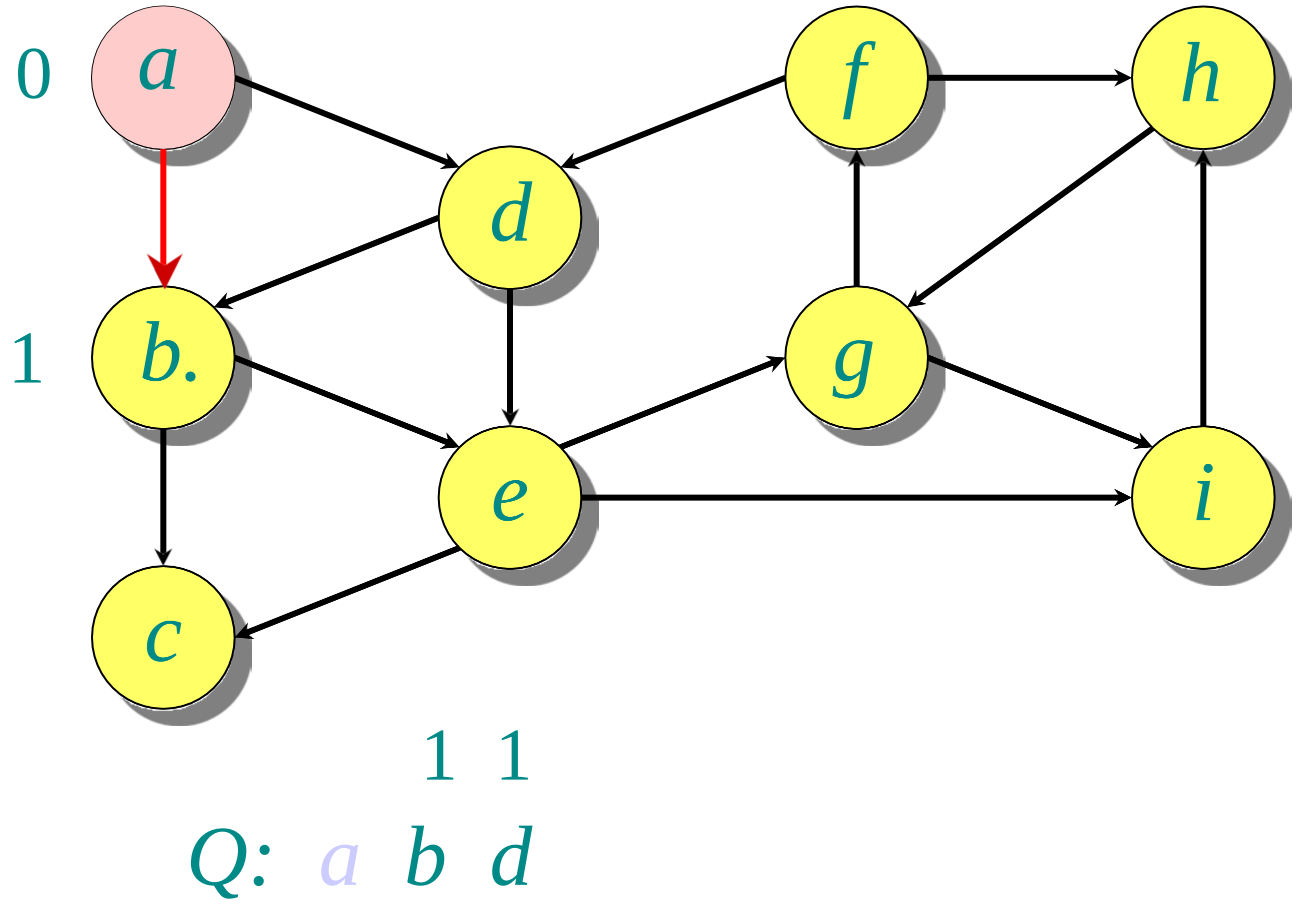


Q:

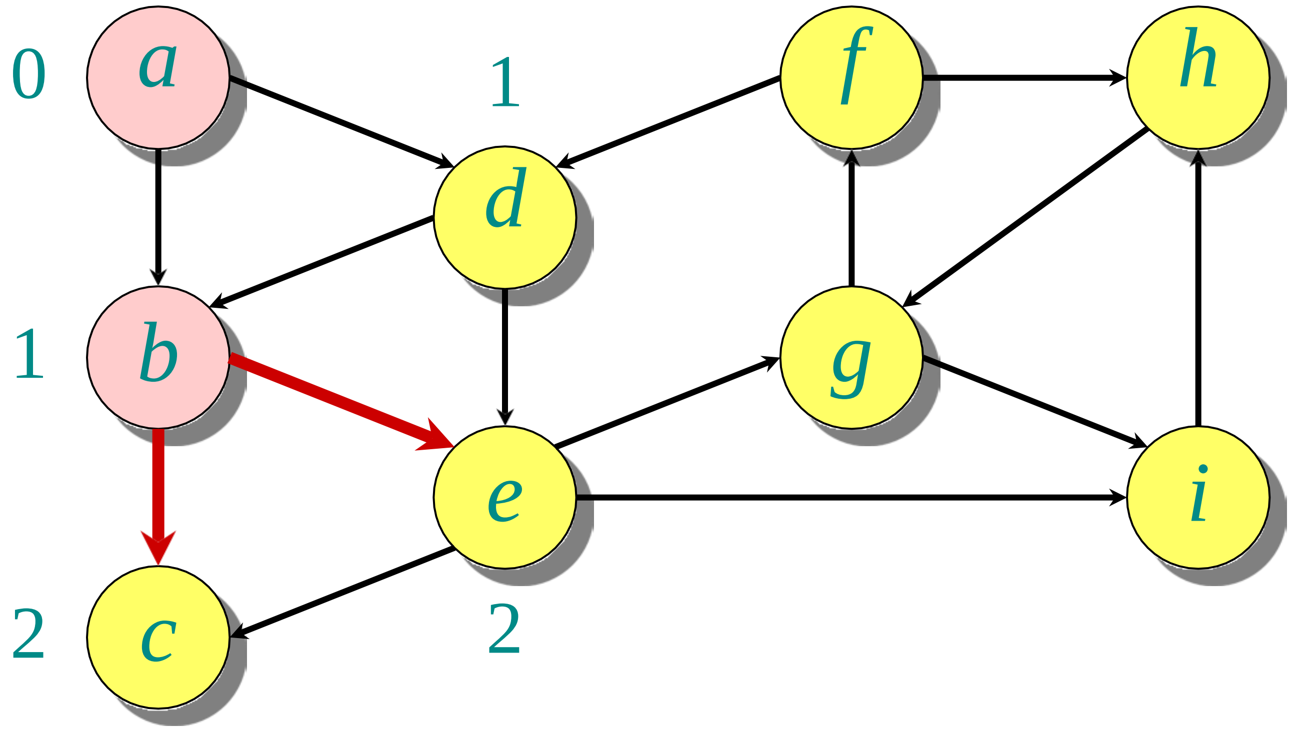
Example of breadth-first search



Example of breadth-first search



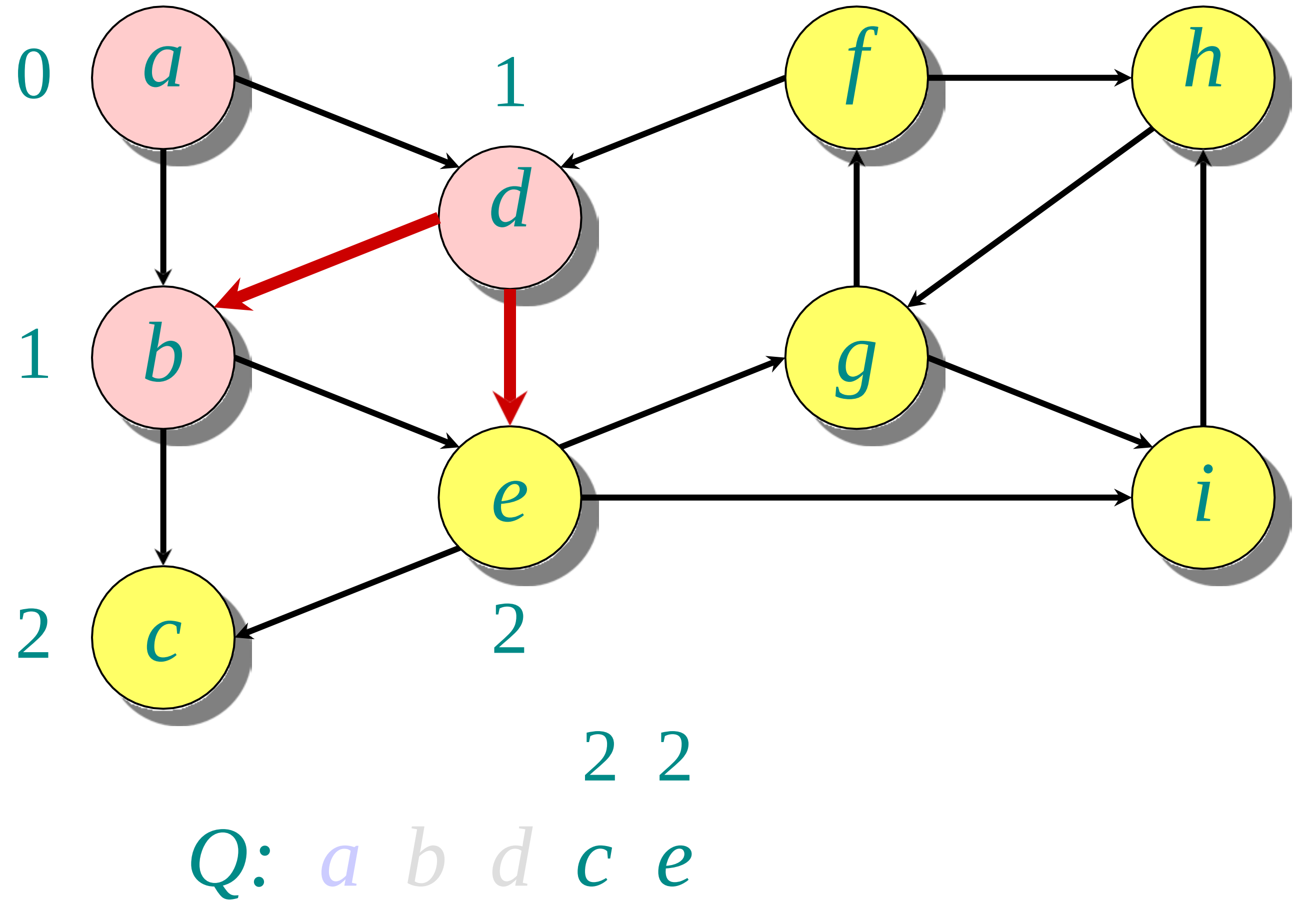
Example of breadth-first search



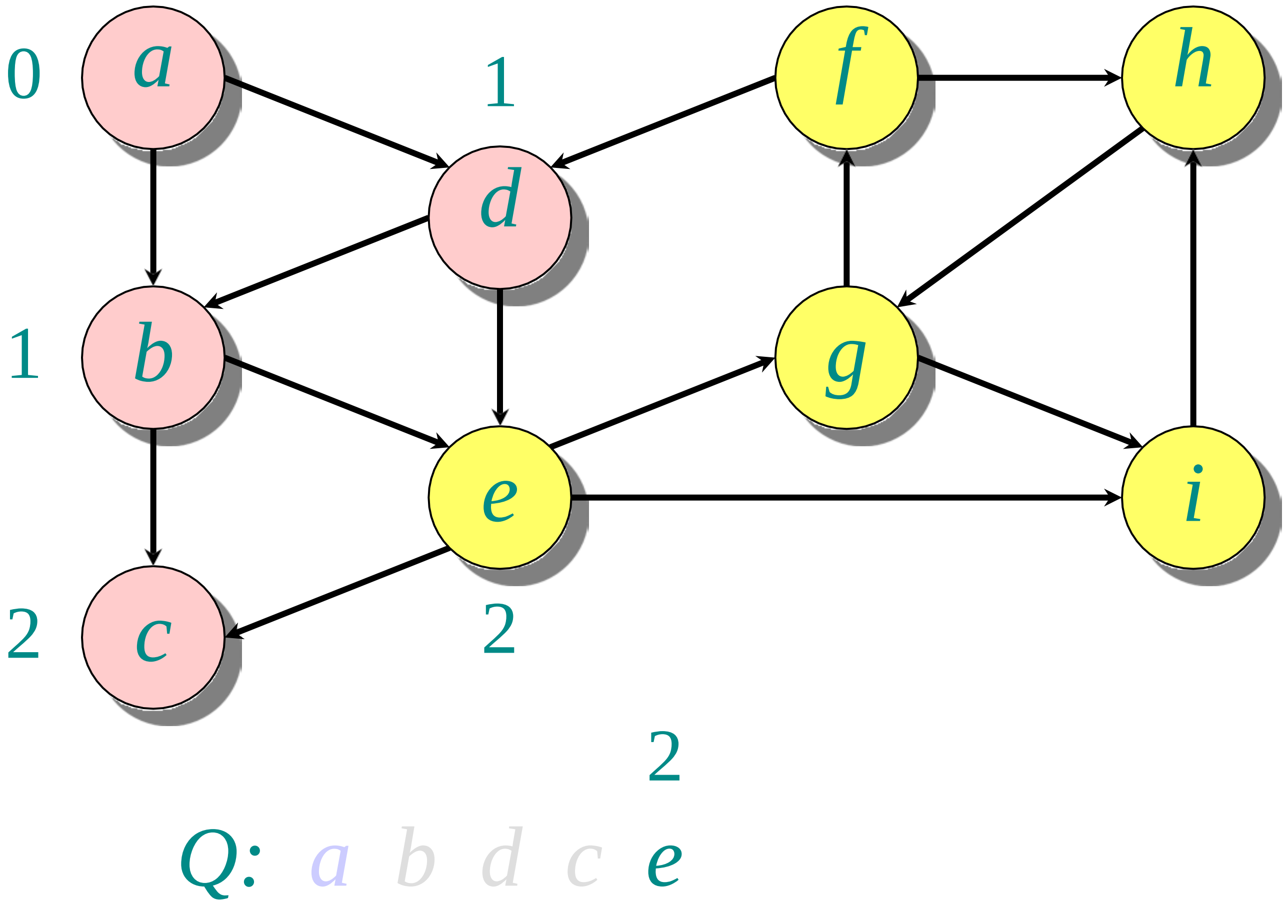
1 2 2

Q: a b d c e

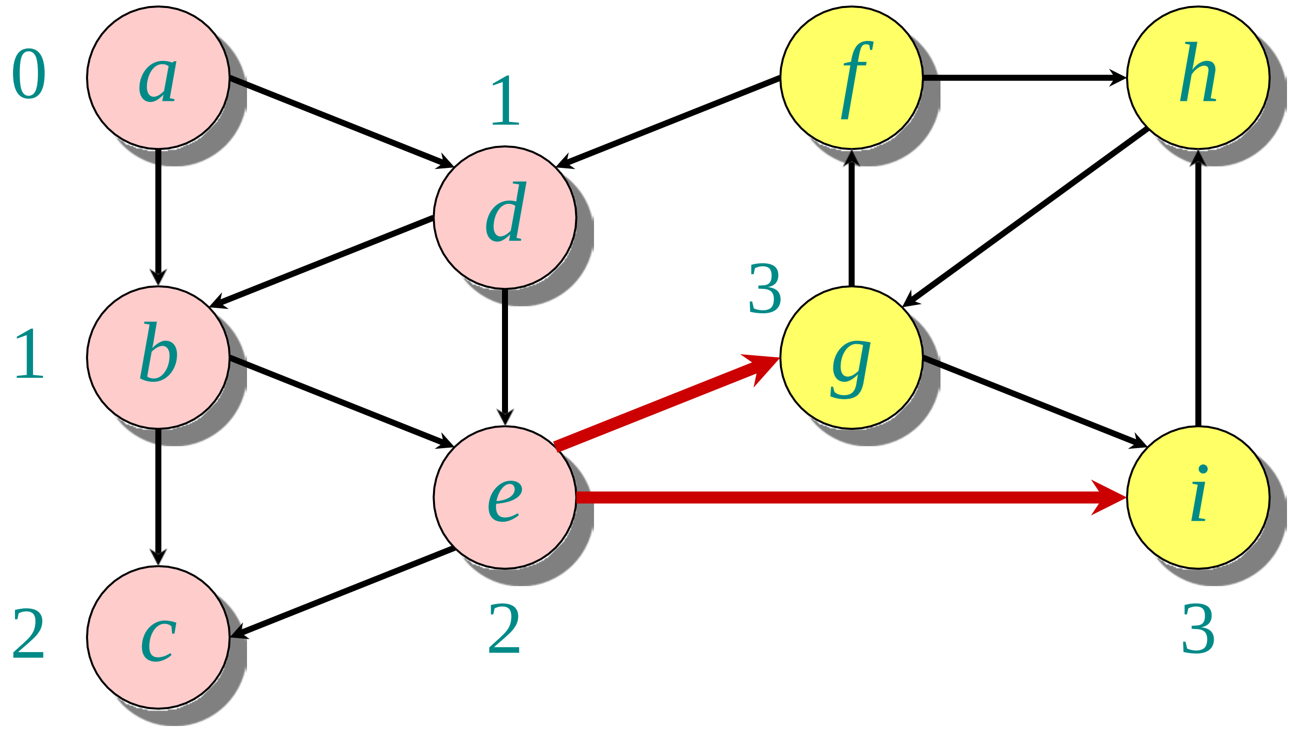
Example of breadth-first search



Example of breadth-first search



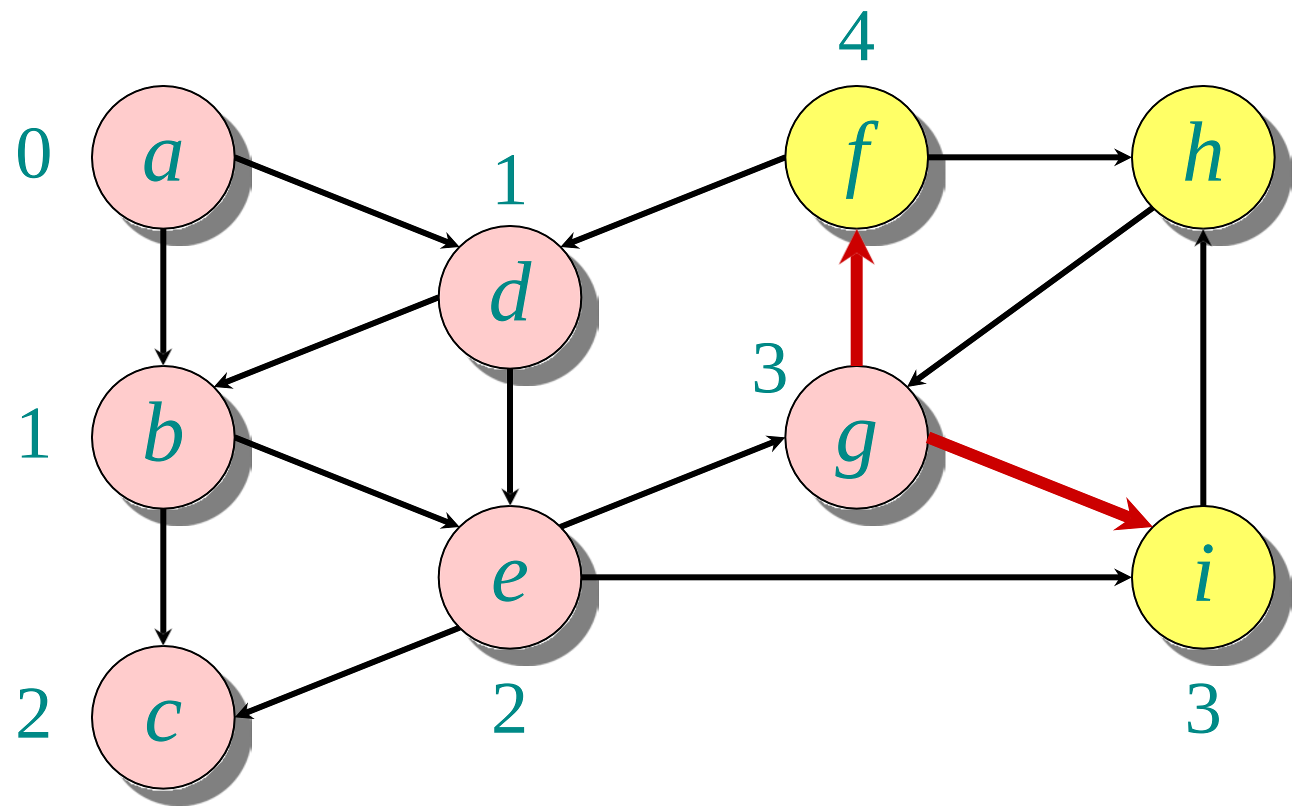
Example of breadth-first search



3 3

Q: a b d c e g i

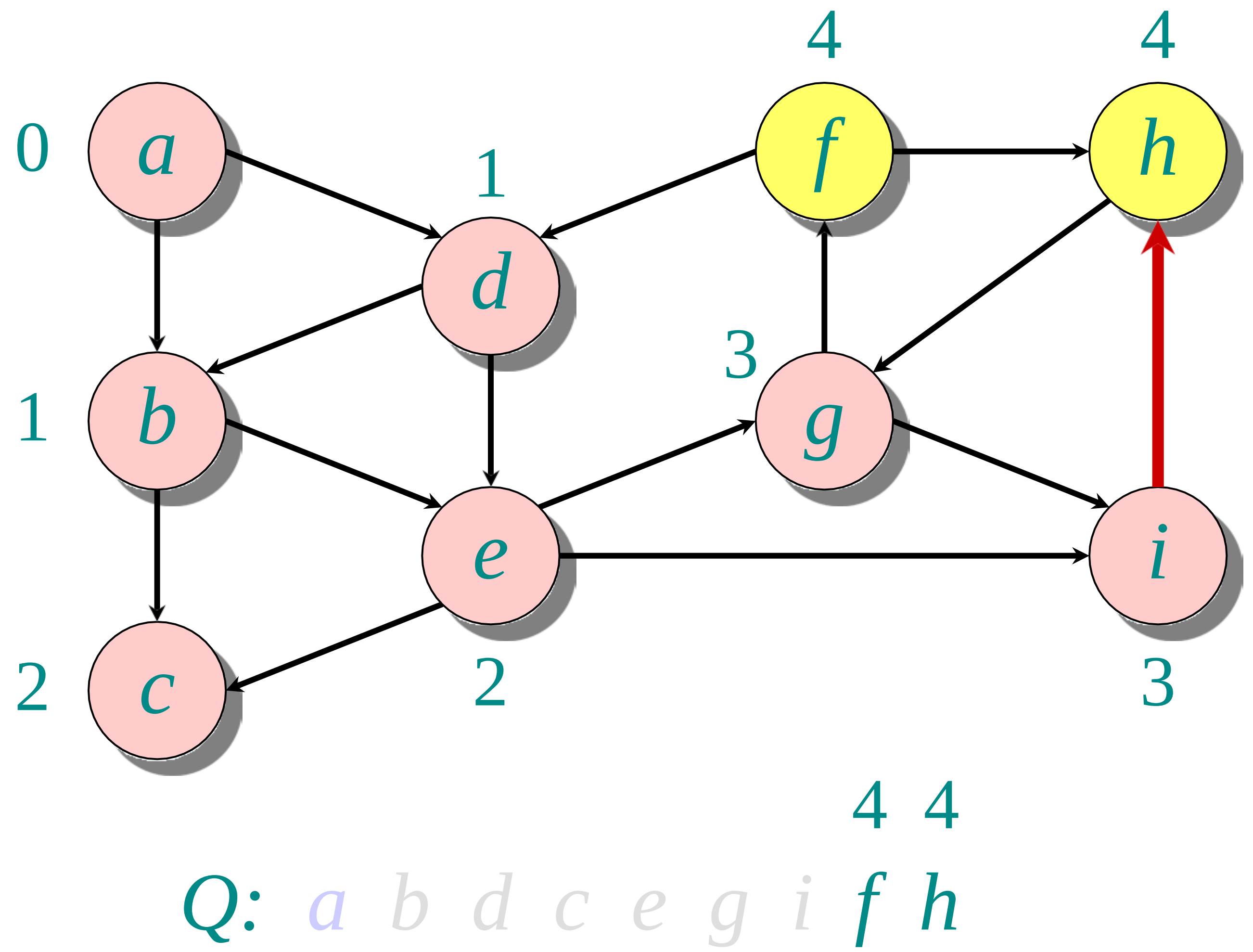
Example of breadth-first search



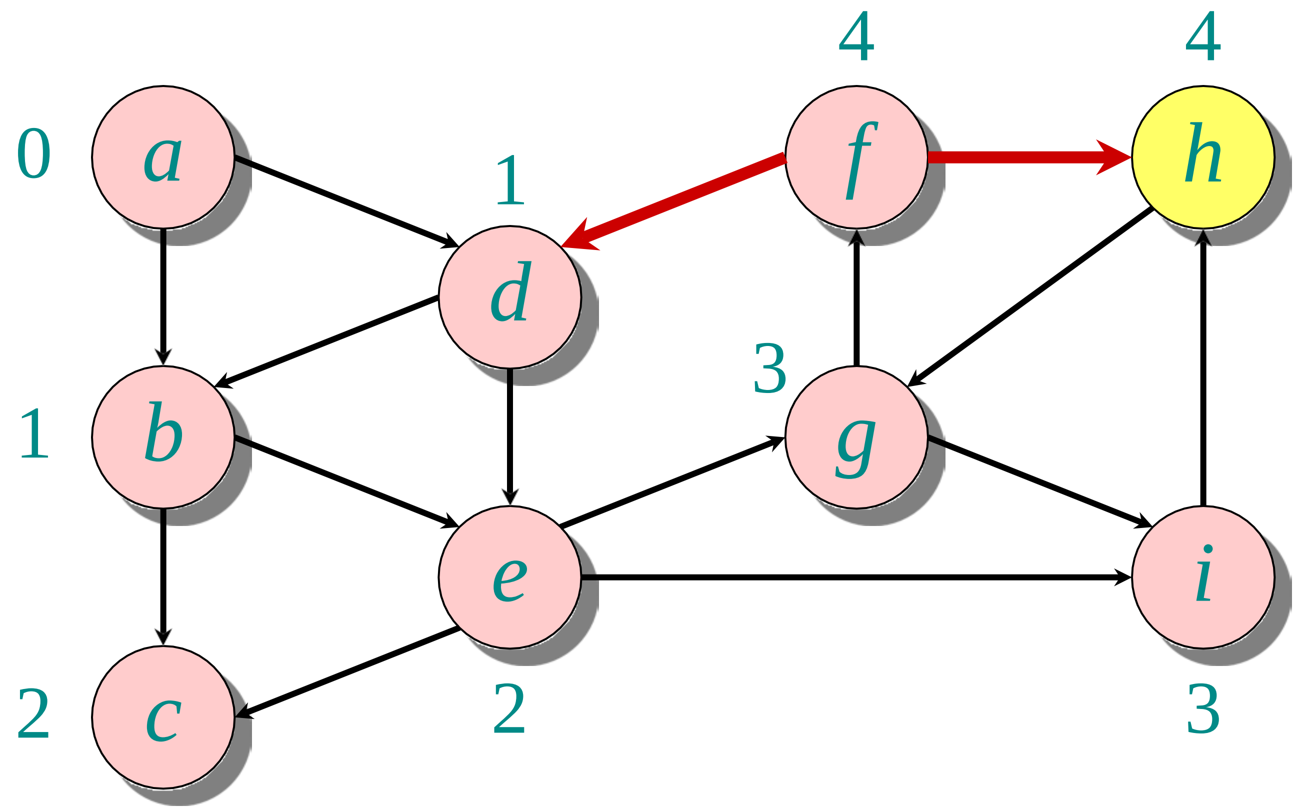
3 4

Q: a b d c e g i f

Example of breadth-first search



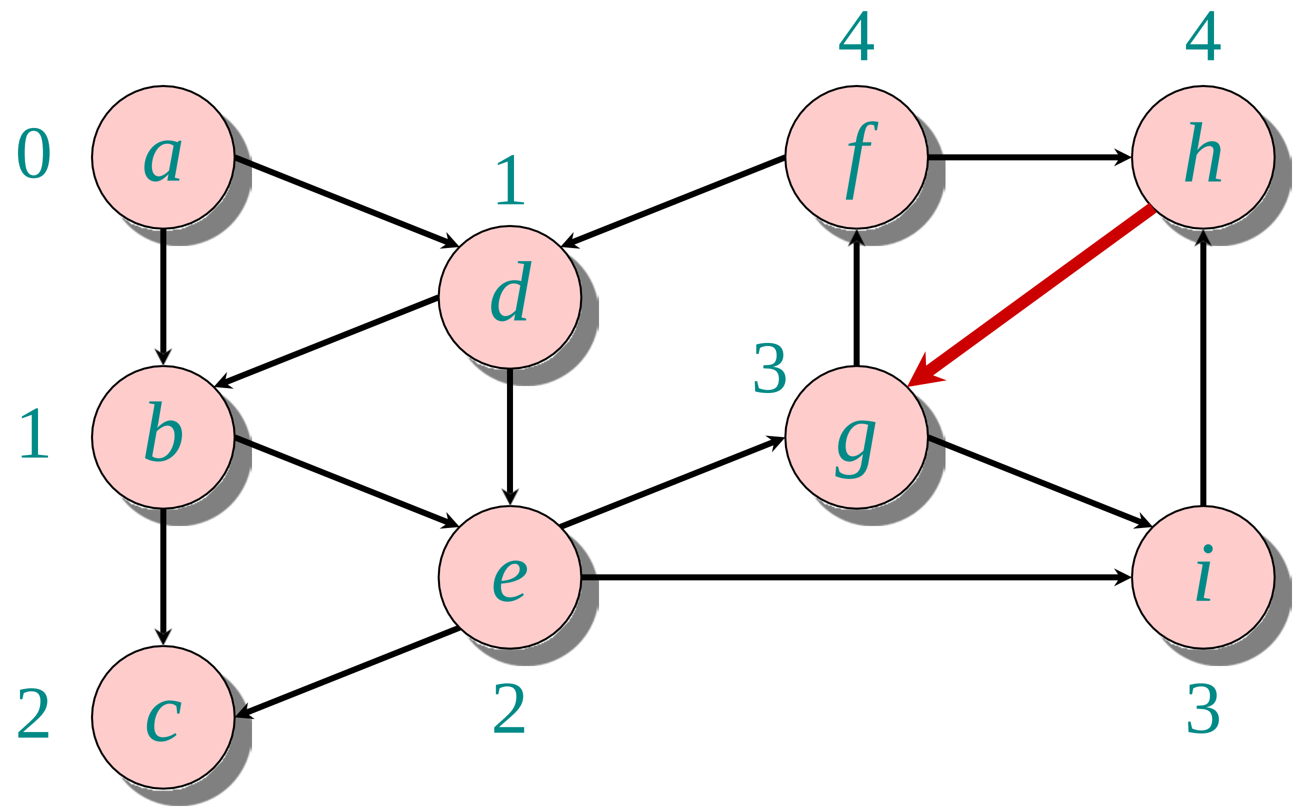
Example of breadth-first search



4

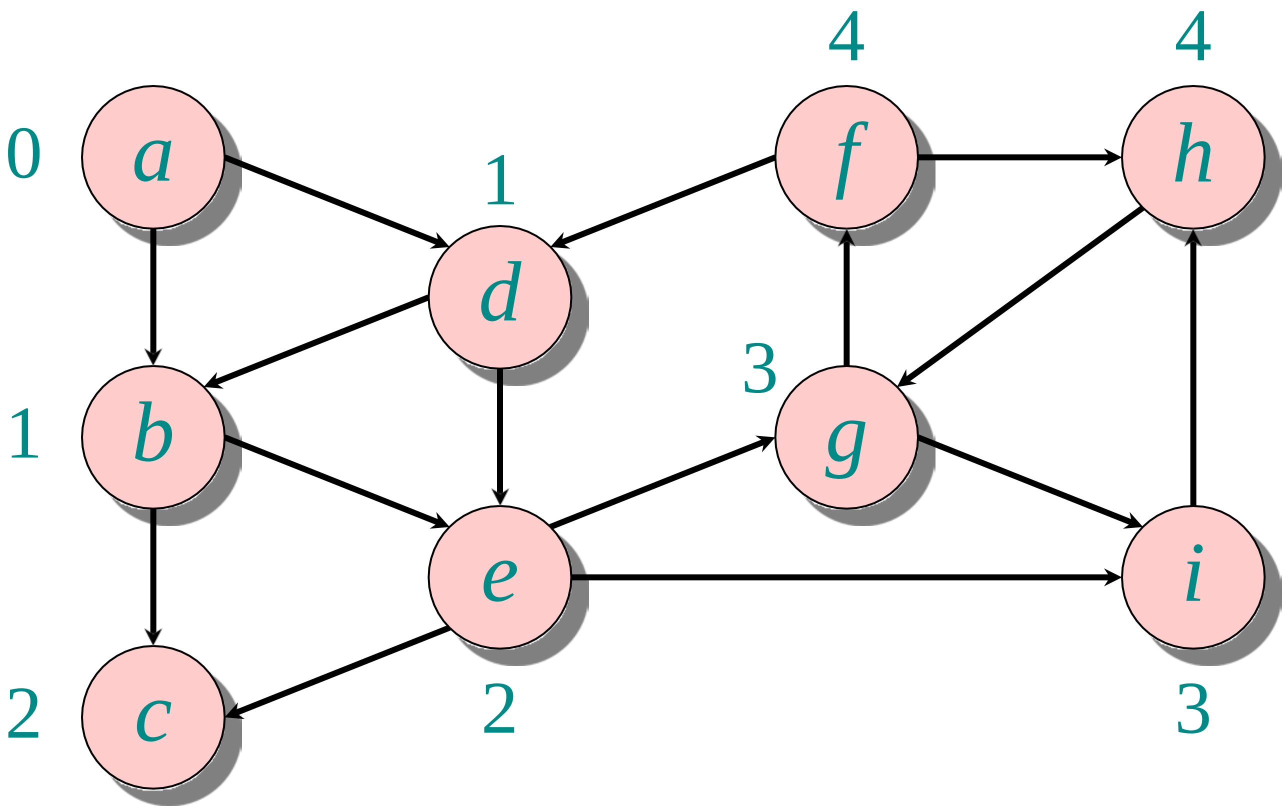
Q: a b d c e g i f h

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

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

Bellman-Ford algorithm

Bellman-Ford algorithm

- (-) Slower than Dijkstra's algorithm
- (+) Can handle negative edge weights.
  - Can be useful if you want to say that some edges are actively good to take, rather than costly.
  - Can be useful as a building block in other algorithms.

Basic idea:

Instead of picking the u with the smallest d[u] to update, just update all of the u's simultaneously.

Bellman-Ford algorithm

Bellman-Ford(G,s):

- d[v] = ∞ for all v in V
- d[s] = 0
- **For** i=0,…,|V|-1:
  - **For** u in V:
    - **For** v in u.neighbors:
      - d[v] ← min(d[v], d[u] + edgeWeight(u,v))

Instead of picking u cleverly,
just update for all of the u's.

Compare to Dijkstra:

- **While** there are **not-sure** nodes:
  - Pick the **not-sure** node u with the smallest estimate **d[u].**
  - **For** v in u.neighbors:
    - d[v] ← min(d[v], d[u] + edgeWeight(u,v))
  - Mark u as **sure**.

Bellman-Ford algorithm

- We are actually going to change this to be less smart.
- Keep n arrays: $d^{(0)}, d^{(1)}, \ldots, d^{(n-1)}$

Bellman-Ford*(G,s):

- $d^{(i)}[v] = \infty$ for all v in V, for all i=0,…,|V|-1
- $d^{(0)}[s] = 0$
- **For** i=0,…,|V|-2:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\,,\, d^{(i+1)}[v],\, d^{(i)}[u] + \mathrm{edgeWeight}(u,v))$
- Then $\mathrm{dist}(s,v) = d^{(n-1)}[v]$

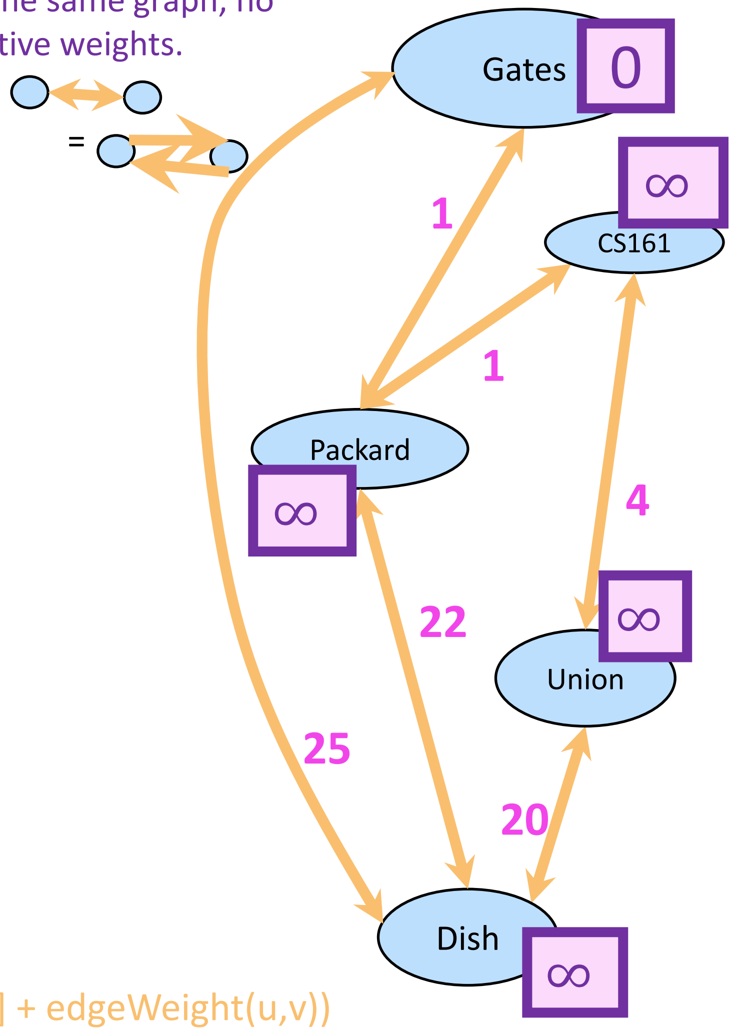
Slightly different than the original Bellman-Ford algorithm, but the analysis is basically the same.

Bellman-Ford

How far is a node from Gates?

	Gates	Packard	CS161	Union	Dish
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$					
$d^{(2)}$					
$d^{(3)}$					
$d^{(4)}$					

Start with the same graph, no negative weights.



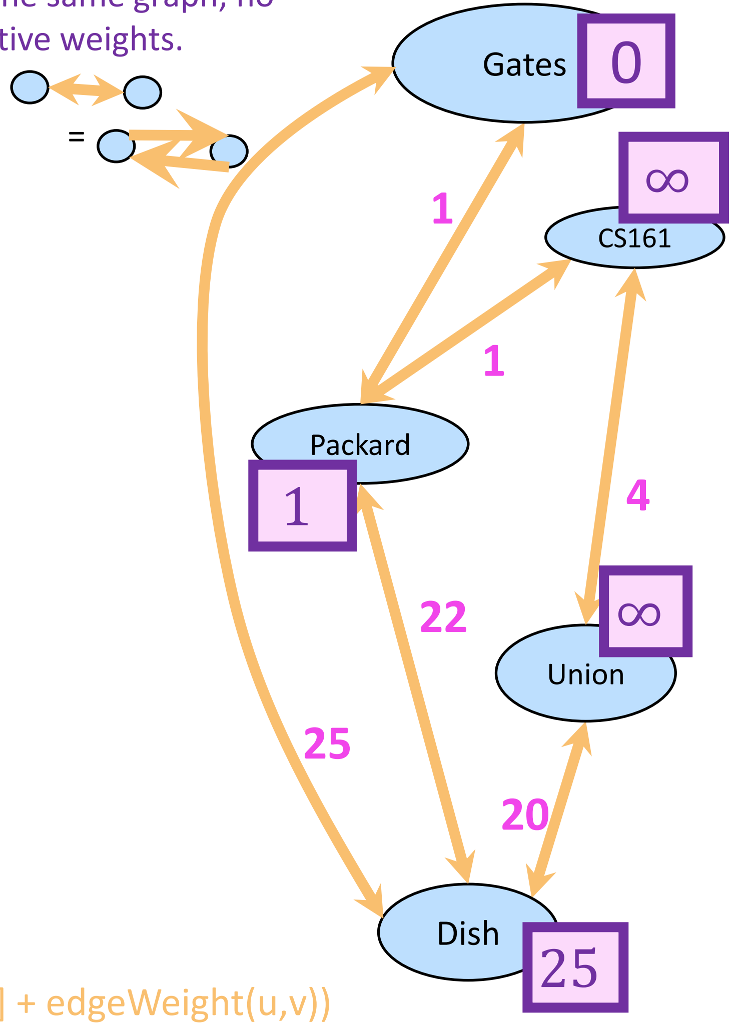
- **For** i=0,…,|V|-2:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\ ,\ d^{(i+1)}[v],\ d^{(i)}[u] + \text{edgeWeight}(u,v))$

Bellman-Ford

How far is a node from Gates?

	Gates	Packard	CS161	Union	Dish
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	1	∞	∞	25
$d^{(2)}$					
$d^{(3)}$					
$d^{(4)}$					

Start with the same graph, no negative weights.



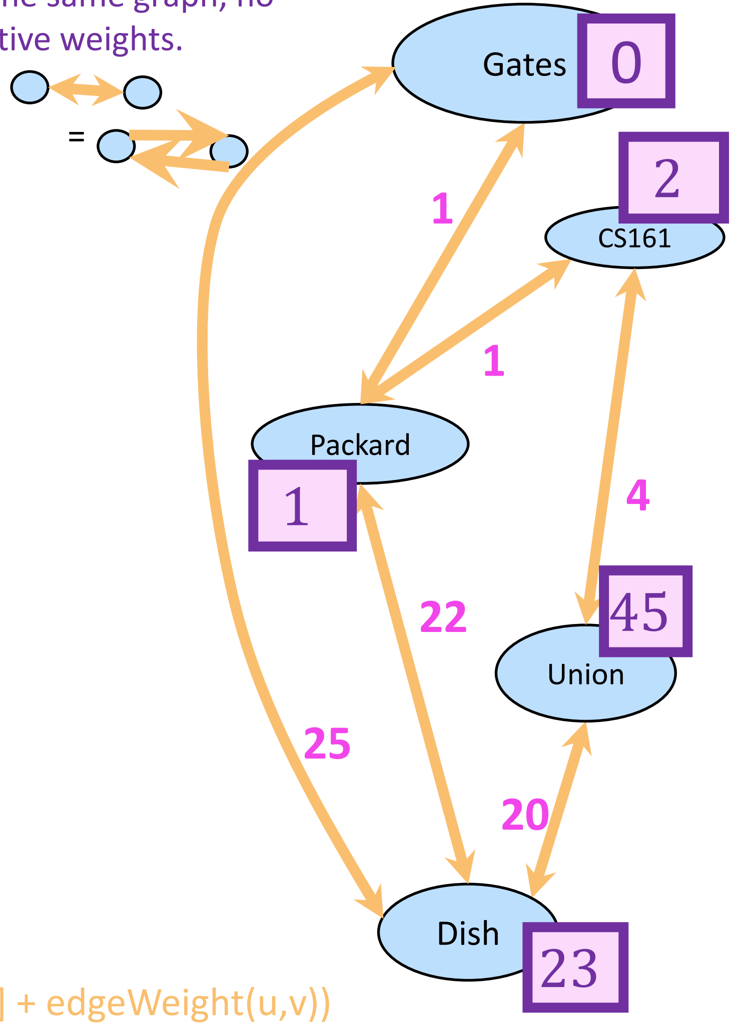
- **For** i=0,…,|V|-2:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\ ,\ d^{(i+1)}[v],\ d^{(i)}[u] + \text{edgeWeight}(u,v))$

Bellman-Ford

How far is a node from Gates?

	Gates	Packard	CS161	Union	Dish
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	1	∞	∞	25
$d^{(2)}$	0	1	2	45	23
$d^{(3)}$					
$d^{(4)}$					

Start with the same graph, no negative weights.



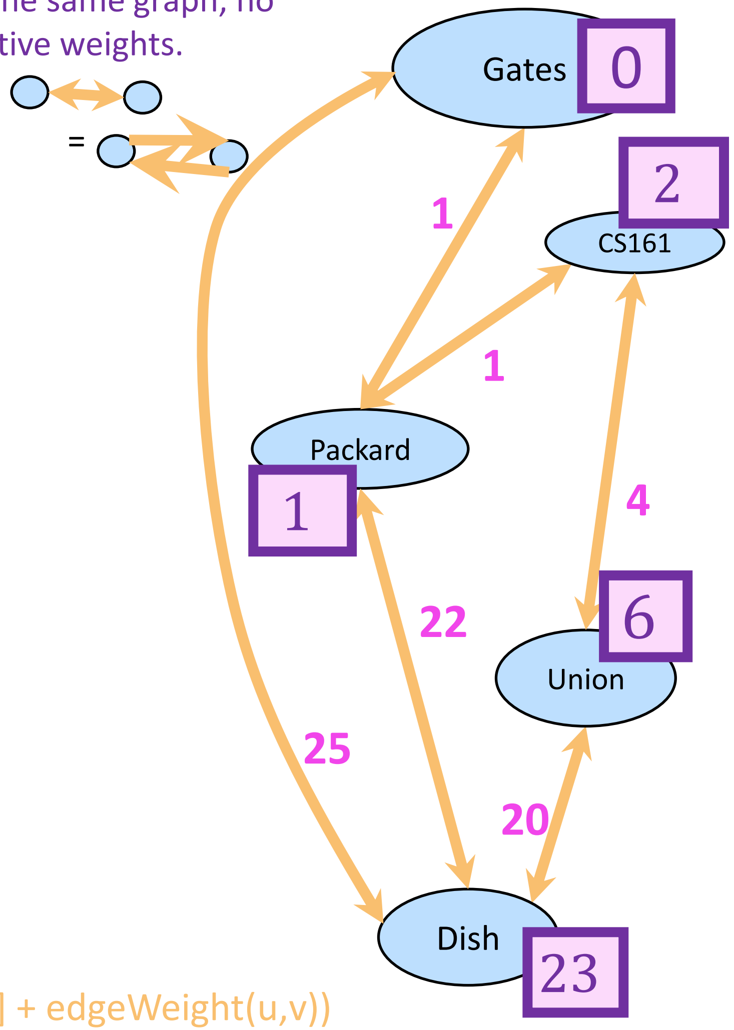
- **For** i=0,…,|V|-2:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v], d^{(i+1)}[v], d^{(i)}[u] + \text{edgeWeight}(u,v))$

Bellman-Ford

How far is a node from Gates?

	Gates	Packard	CS161	Union	Dish
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	1	∞	∞	25
$d^{(2)}$	0	1	2	45	23
$d^{(3)}$	0	1	2	6	23
$d^{(4)}$					

Start with the same graph, no negative weights.



- **For** i=0,…,|V|-2:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\ ,\ d^{(i+1)}[v],\ d^{(i)}[u] + \mathrm{edgeWeight}(u,v))$

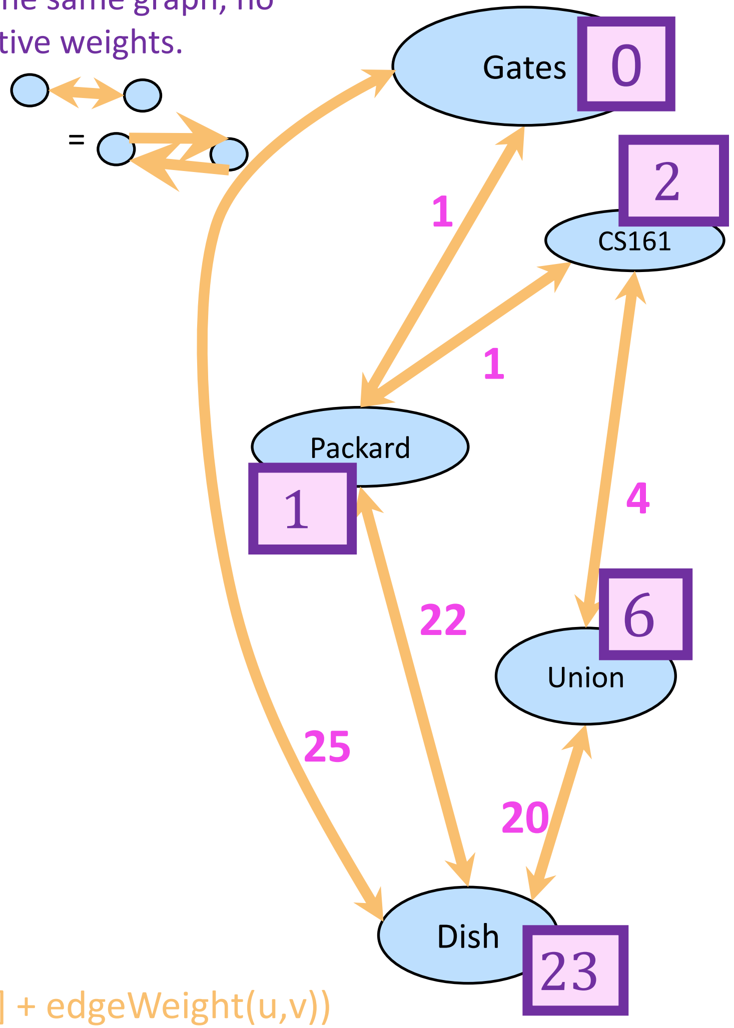
Bellman-Ford

How far is a node from Gates?

	Gates	Packard	CS161	Union	Dish
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	1	∞	∞	25
$d^{(2)}$	0	1	2	45	23
$d^{(3)}$	0	1	2	6	23
$d^{(4)}$	0	1	2	6	23

These are the final distances!

Start with the same graph, no negative weights.



- **For** i=0,…,|V|-2:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v] , d^{(i+1)}[v], d^{(i)}[u] + edgeWeight(u,v))$

Bellman-Ford

- Does it work?
  - Yes
  - Idea to the right.

	Gates	Packard	CS161	Union	Dish
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**25**
$d^{(2)}$	0	1	**2**	**45**	**23**
$d^{(3)}$	0	1	2	**6**	23
$d^{(4)}$	0	1	2	6	23

- Is it fast?
  - Not really…

Inductive Hypothesis:
$d^{(i)}[v]$ is equal to the cost of the shortest path between s and v **with at most i edges**.
Conclusion:
$d^{(|V|-1)}[v]$ is equal to the cost of the shortest simple path between s and v. **(Since all simple paths have at most |V|-1 edges).**

A **simple path** is a path with no cycles.

Proof by induction

- **Inductive Hypothesis:**
  - After iteration i, for each v, $d^{(i)}[v]$ is equal to the cost of the shortest path between s and v with at most i edges.
- **Base case:**
  - After iteration 0… ✓
- **Inductive step:**

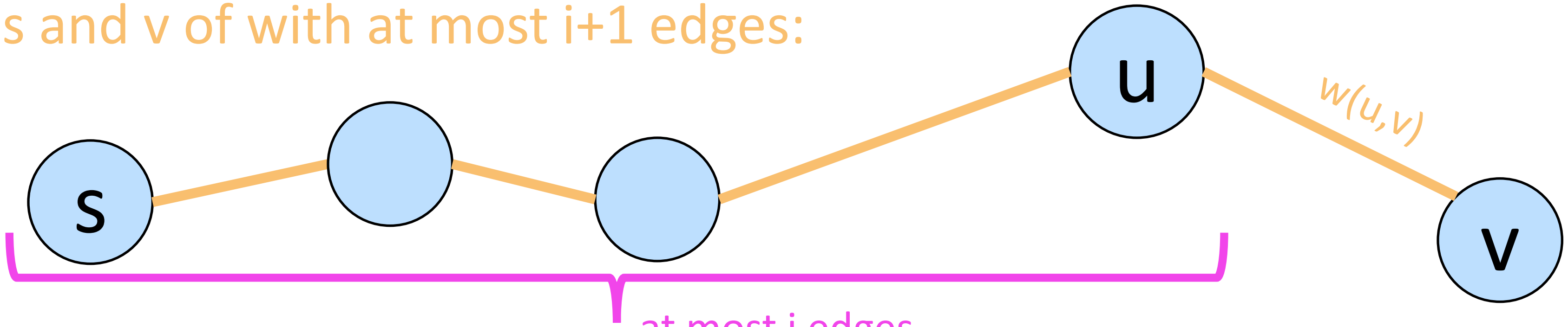
Inductive step

- Suppose the inductive hypothesis holds for i.
- We want to establish it for i+1.

Hypothesis: After iteration i, for each v, $d^{(i)}[v]$ is equal to the cost of the shortest path between s and v with at most i edges.

Say this is the shortest path between s and v of with at most i+1 edges:

Let u be the vertex right before v in this path.



- By induction, $d^{(i)}[u]$ is the cost of a shortest path between s and u of i edges.
- By setup, $d^{(i)}[u] + w(u,v)$ is the cost of a shortest path between s and v of i+1 edges.
- In the i+1'st iteration, we ensure $\mathbf{d^{(i+1)}[v] <= d^{(i)}[u] + w(u,v)}$.
- So $d^{(i+1)}[v]$ <= cost of shortest path between s and v with i+1 edges.
- But $d^{(i+1)}[v]$ = cost of a particular path of at most i+1 edges >= cost of shortest path.
- So d[v] = cost of shortest path with at most i+1 edges.

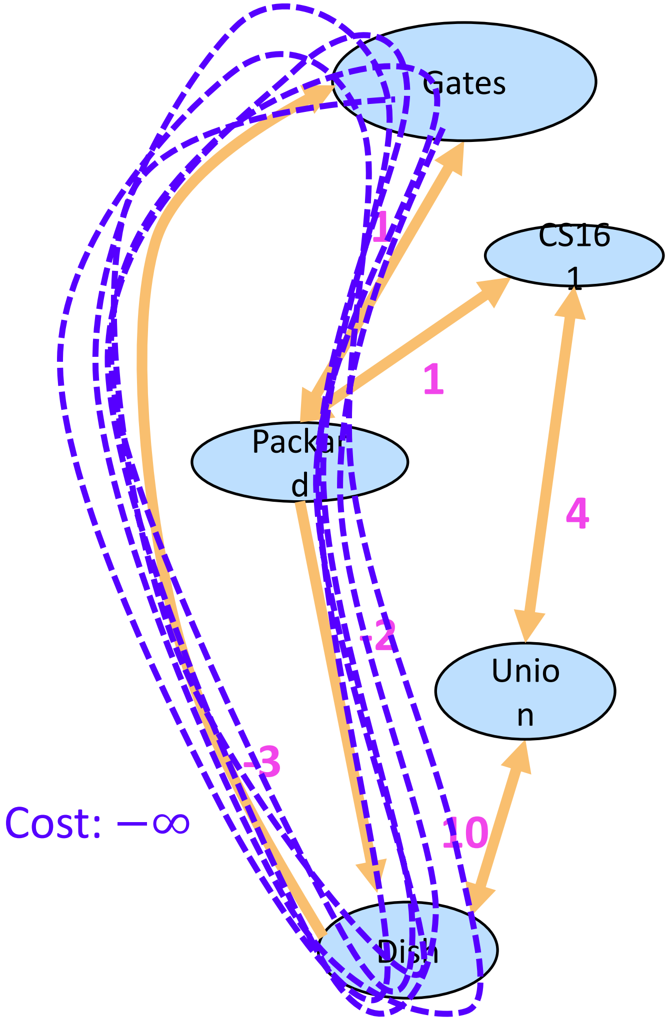
Pros and cons of Bellman-Ford

- Running time: $O(|V||E|)$ running time
  - For each of |V| steps we update m edges
  - Slower than Dijkstra
- However, it's also more flexible in a few ways.
  - Can handle negative edges
  - If we constantly do these iterations, any changes in the network will eventually propagate through.

Negative edge weights?

- What is the shortest path from Gates to the Union?
- Shortest paths aren't defined if there are negative cycles!

- B-F works with negative edge weights…as long as there are not negative cycles.
  - A negative cycle is a path with the same start and end vertex whose cost is negative.
- However, B-F can detect negative cycles.



How Bellman-Ford deals with negative cycles

- If there are no negative cycles:
  - Everything works as it should.
  - The algorithm stabilizes after |V|-1 rounds.
  - Note: Negative ***edges*** are okay!!
- If there are negative cycles:
  - Not everything works as it should…
    - it couldn't possibly work, since shortest paths aren't well-defined if there are negative cycles.
  - The d[v] values will keep changing.
- Solution:
  - Go one round more and see if things change.
    - If so, return NEGATIVE CYCLE ☹

Summary

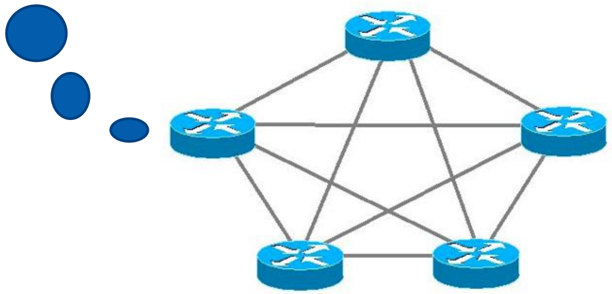
- The Bellman-Ford algorithm:
  - –Finds shortest paths in weighted graphs with negative edge weights
  - –runs in time O(|V||E|) on a graph G with n vertices and m edges.
- If there are no negative cycles in G:
  - –the BF algorithm terminates with $d^{(|V|-1)}[v] = d(s,v)$.
- If there are negative cycles in G:
  - –the BF algorithm returns `negative cycle`.

Bellman-Ford is also used in practice.

- eg, Routing Information Protocol (RIP) uses something like Bellman-Ford.
  - –Older protocol, not used as much anymore.

- Each router keeps a **table** of distances to every other router.
- Periodically we do a Bellman-Ford update.
- This means that if there are changes in the network, this will propagate. (maybe slowly…)

Destination	Cost to get there	Send to whom?
172.16.1.0	34	172.16.1.1
10.20.40.1	10	192.168.1.2
10.155.120.1	9	10.13.50.0



Single-source shortest paths

- Nonnegative edge weights
  - Dijkstra's algorithm: $O(|E| + |V| \lg|V|)$
- General
  - Bellman-Ford algorithm: $O(|V||E|)$

All-pairs shortest paths

- Nonnegative edge weights
  - Dijkstra's algorithm $|V|$ times: $O(|V||E| + |V|^2 \lg|V|)$
- General
  - Floyd-Warshall algorithms: $\Theta(|V|^3)$.

Remain for Dynamic Programming

The End