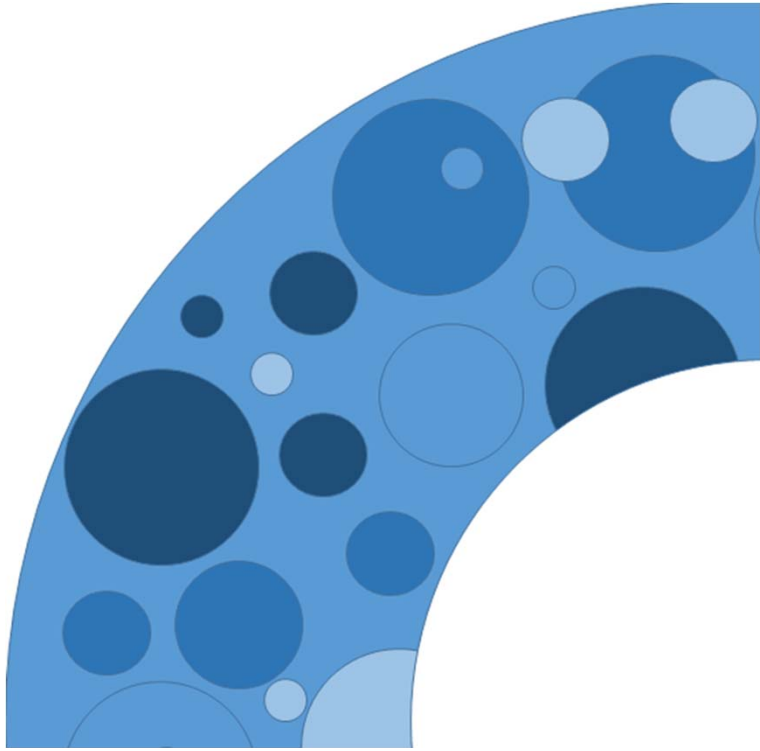
Algorithms

Ok-Ran Jeong

School of Computing, Gachon University



5. Graph Algorithms II
(SSSP: Single-Source Shortest Paths)

Contents

- Paths in graphs
- Single-source shortest paths
- Dijkstra's algorithm
- Bellman-Ford algorithm

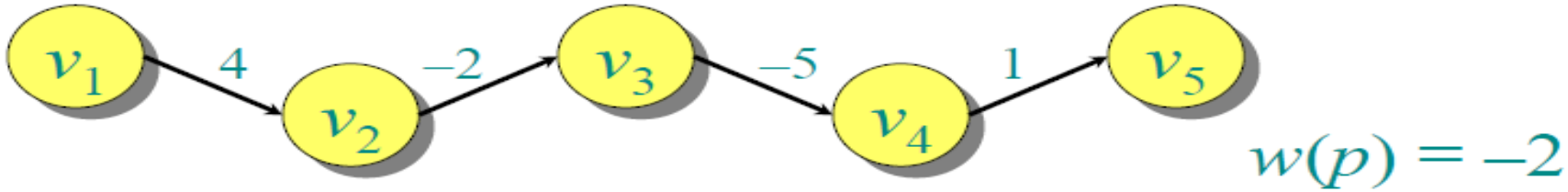
- Problem 10: Contest trip

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

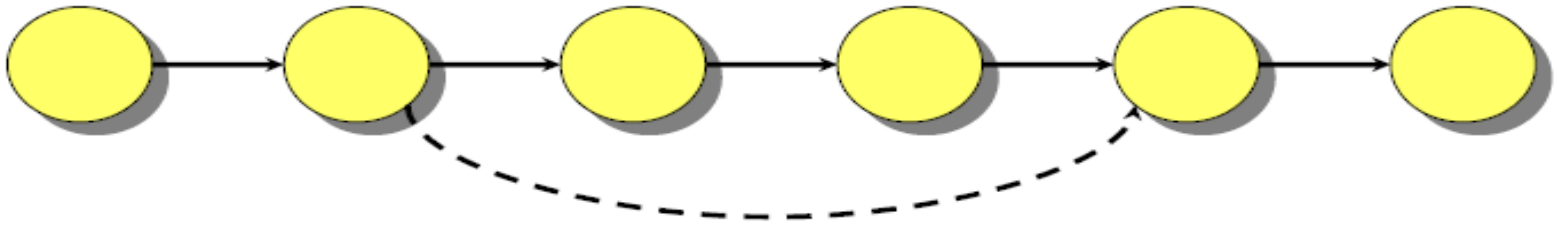
$\delta(u, v) = \min\{w(p) : p \text{ is a path from } u \text{ to } v\}$.

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

Optimal substructure

Theorem. A sub-path of a shortest path is a shortest path.

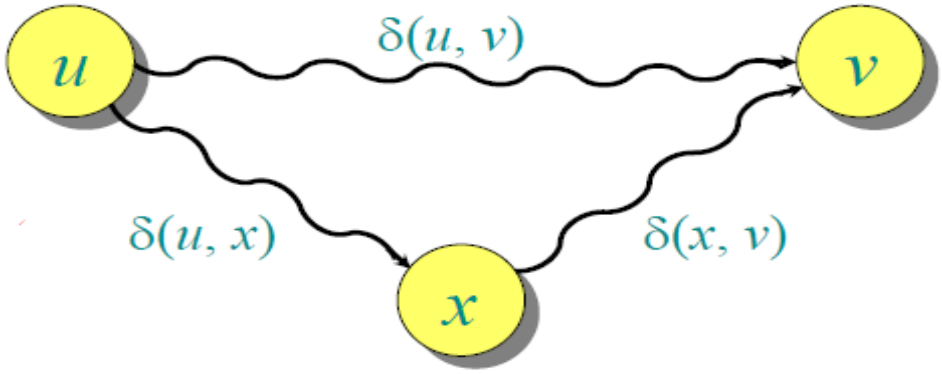
Proof. Cut and paste:



Triangle inequality

Theorem. For all $u, v, x \in V$, we have
$\delta(u, v) \leq \delta(u, x) + \delta(x, v)$.

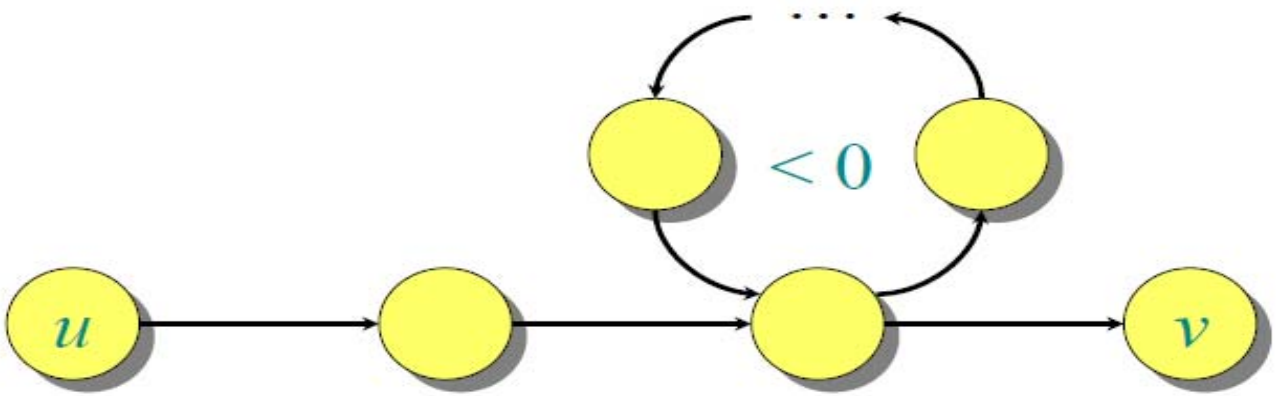
Proof.



Well-definedness of shortest paths

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

Example:



Single-source shortest paths

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

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

IDEA: Greedy.

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

Dijkstra's algorithm

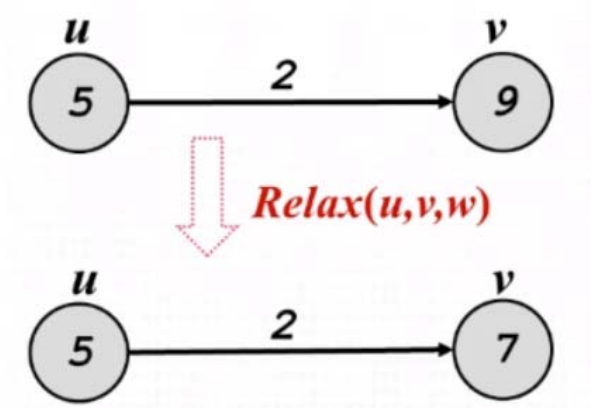
```
1  d[s] ← 0
2  for each v ∈ V − {s}
3       do d[v] ← ∞
4  S ← ∅
5  Q ← V            ▷ Q is a priority queue maintaining V − S
6  while Q ≠ ∅
7       do u ← EXTRACT-MIN(Q)
8          S ← S ∪ {u}
9          for each v ∈ Adj[u]
10             do if d[v] > d[u] + w(u, v)
11                  then d[v] ← d[u] + w(u, v)
```

relaxation step

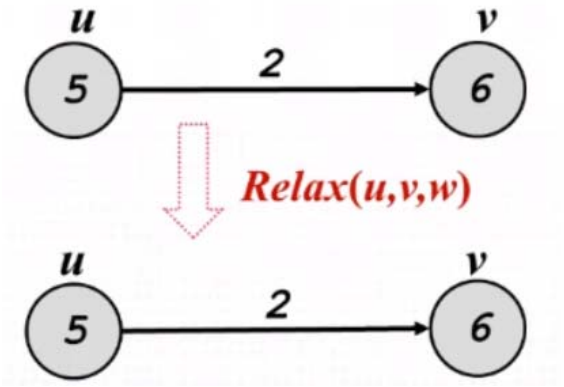
DECREASE-KEY

Relaxation

- Examples for Relaxation



(a)



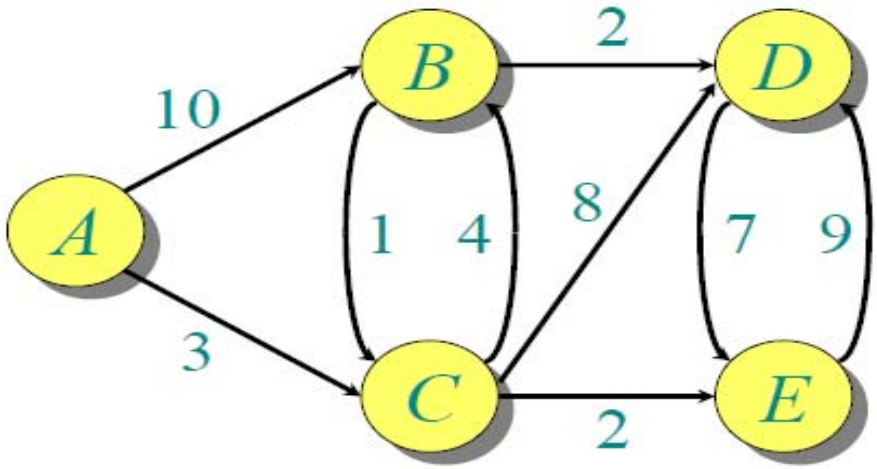
(b)

```
for each v ∈ Adj[u]
    do if d[v] > d[u] + w(u, v)
        then d[v] ← d[u] + w(u, v)
```

relaxation step

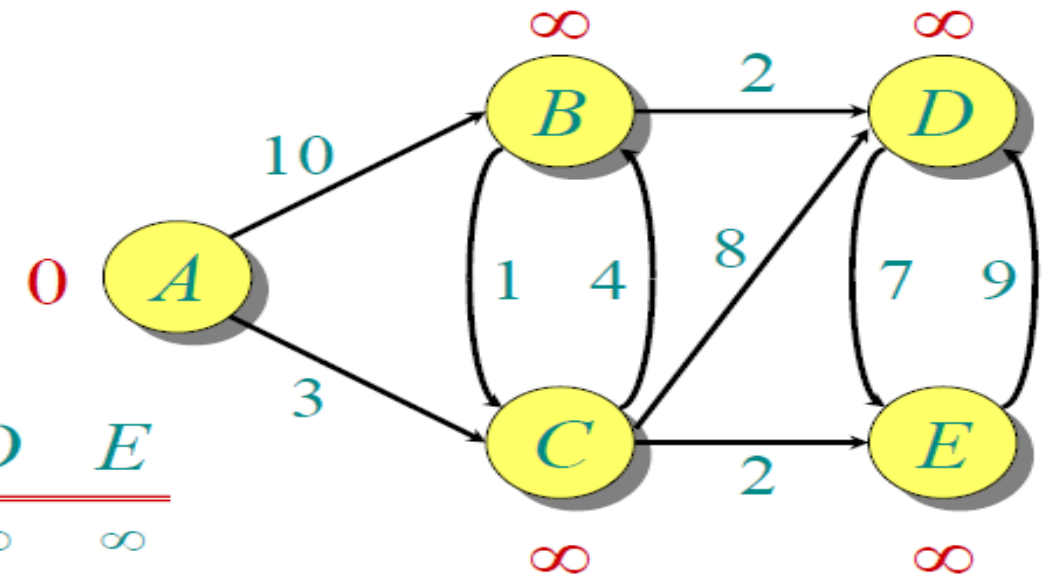
Example of Dijkstra's algorithm

Graph with nonnegative edge weights:



Example of Dijkstra's algorithm

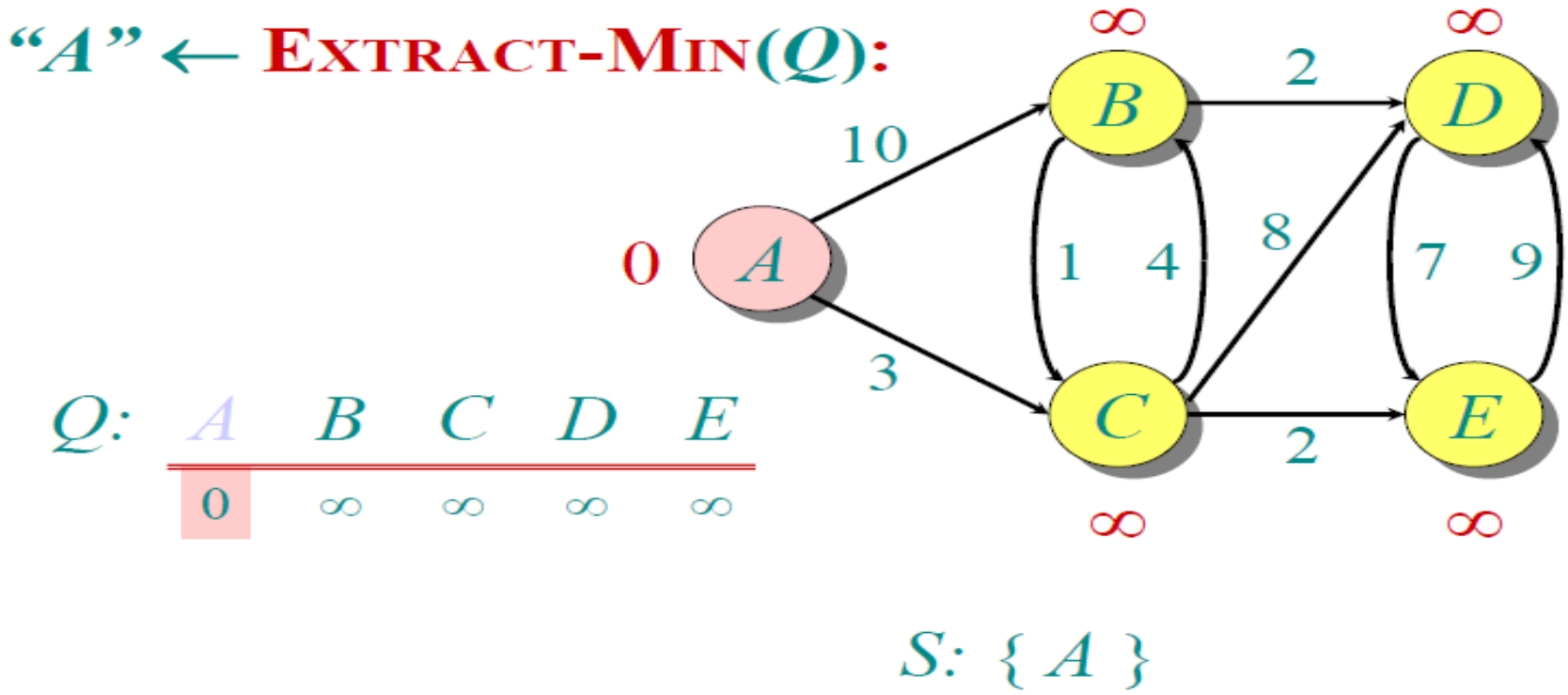
Initialize:



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

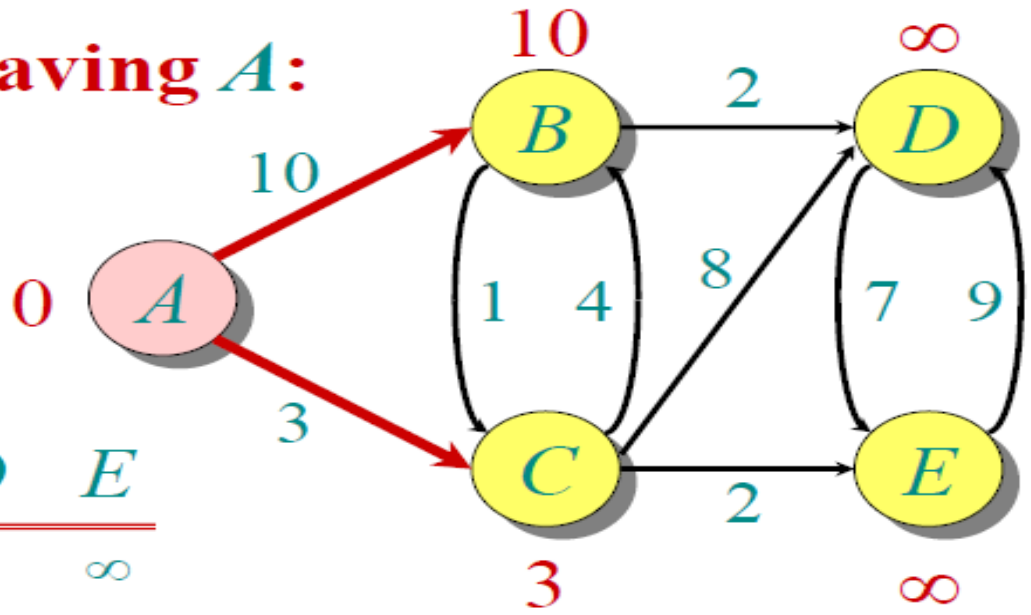
S: {}

Example of Dijkstra's algorithm



Example of Dijkstra's algorithm

Relax all edges leaving A:

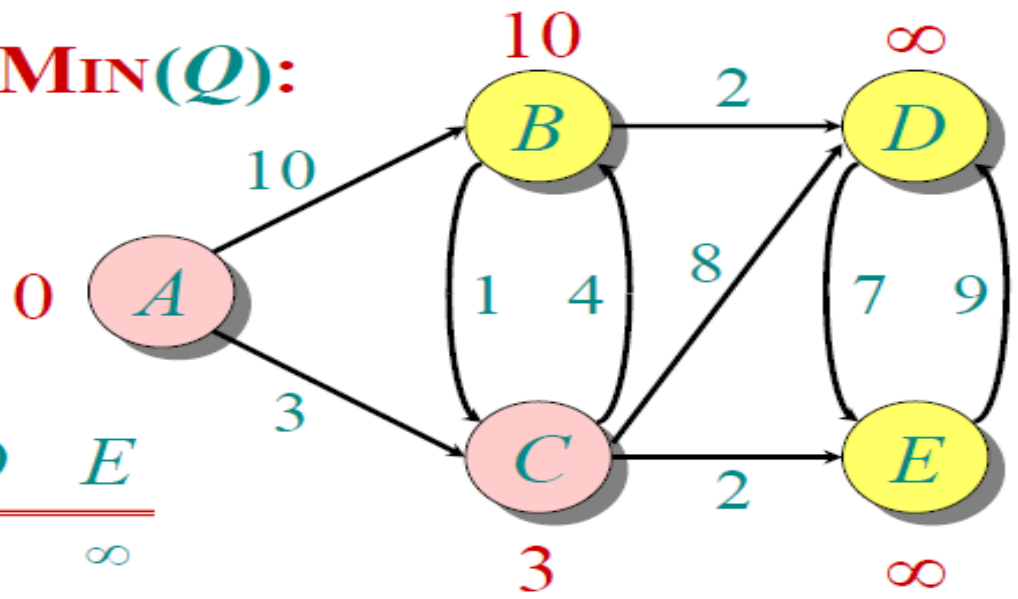


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	—	—

S: { A }

Example of Dijkstra's algorithm

"C" ← EXTRACT-MIN(Q):

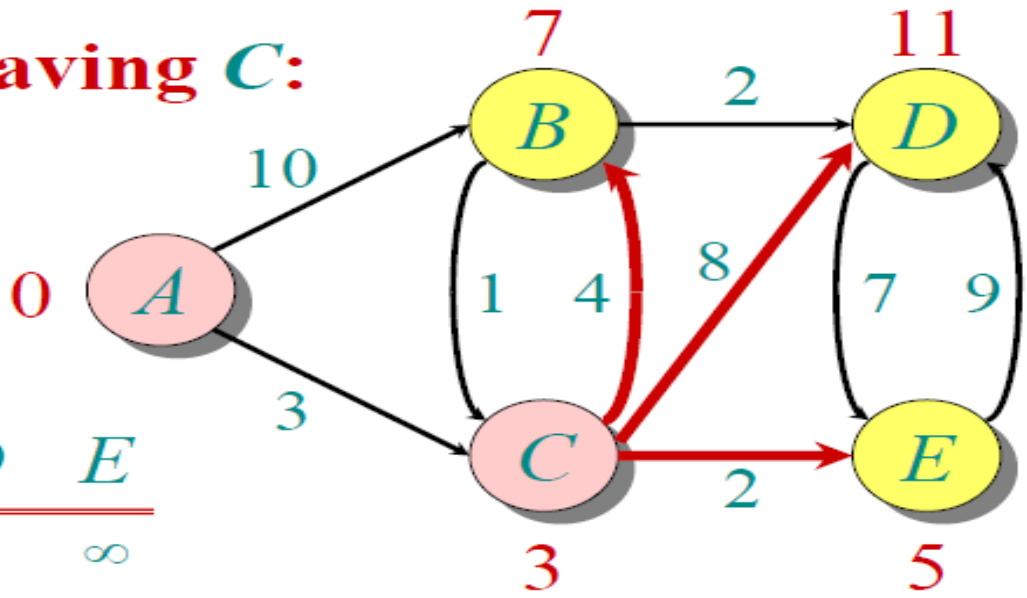


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	—	—

S: { A, C }

Example of Dijkstra's algorithm

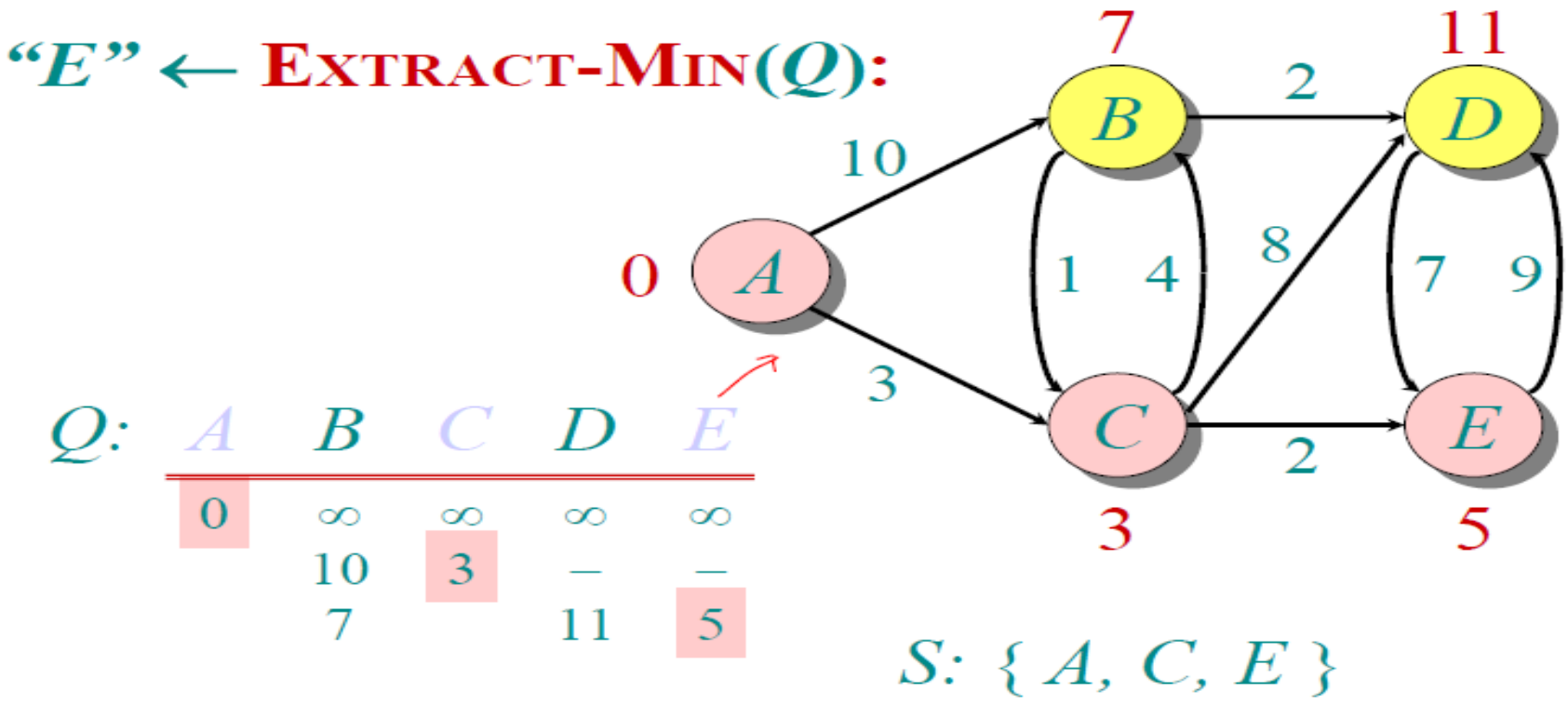
Relax all edges leaving C:



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

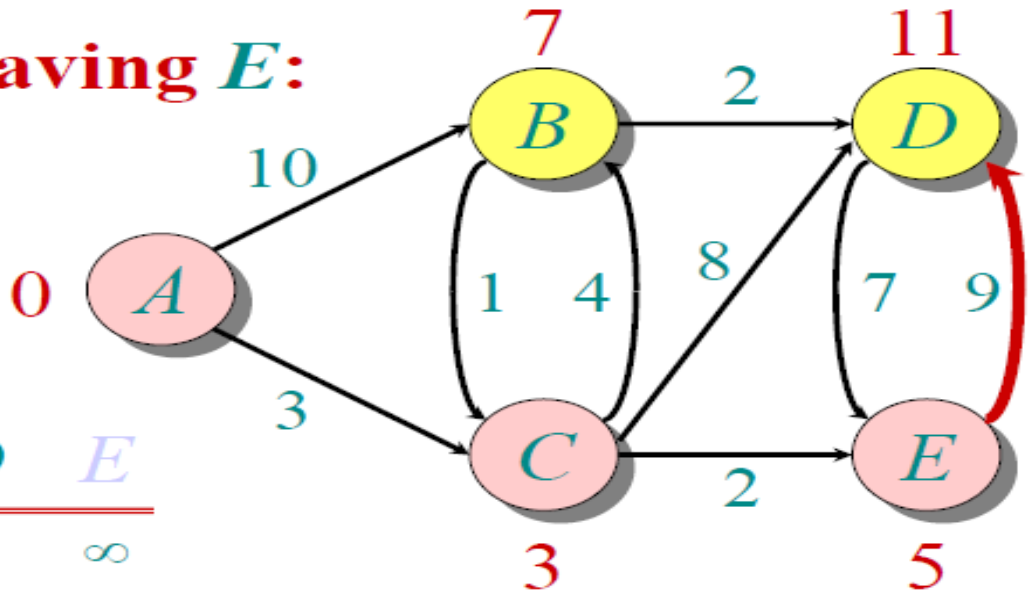
S: { A, C }

Example of Dijkstra's algorithm



Example of Dijkstra's algorithm

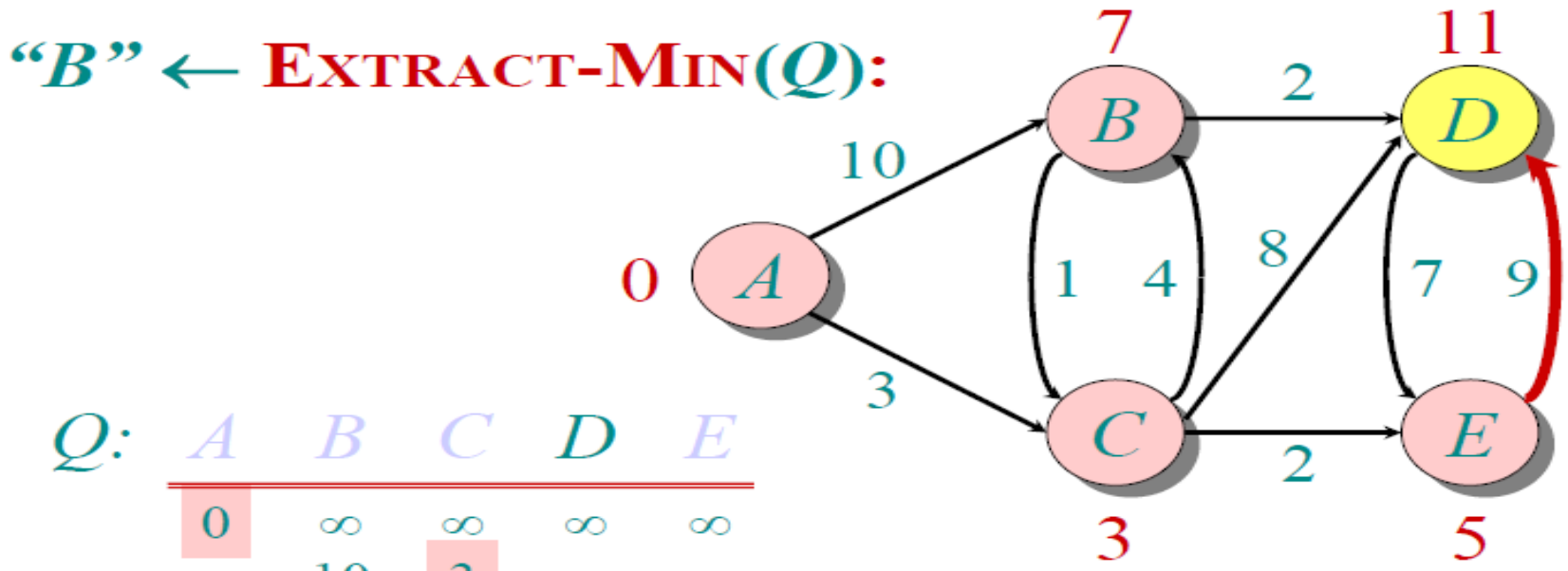
Relax all edges leaving E:



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

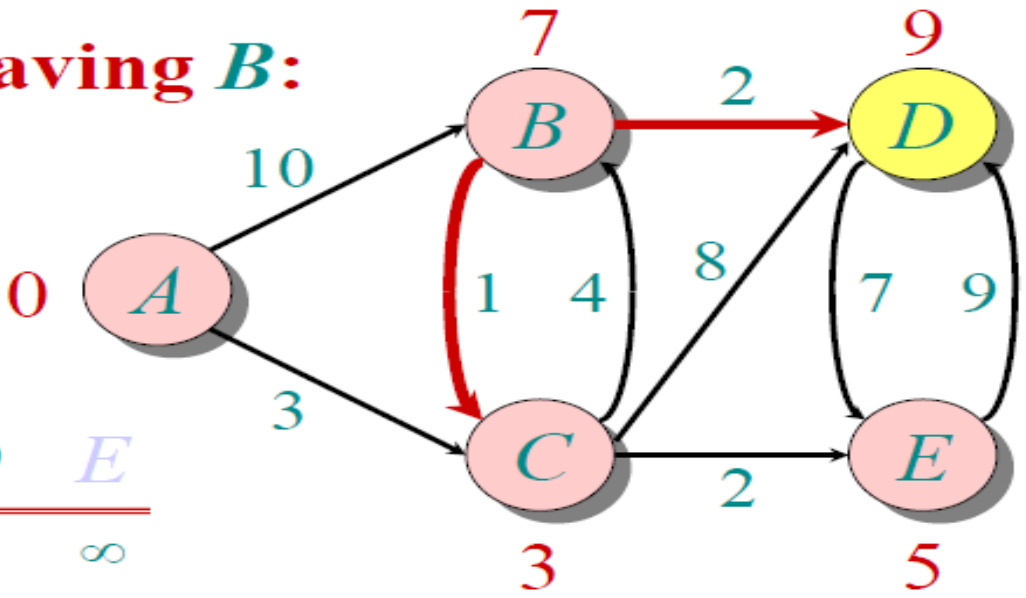
S: { A, C, E }

Example of Dijkstra's algorithm



Example of Dijkstra's algorithm

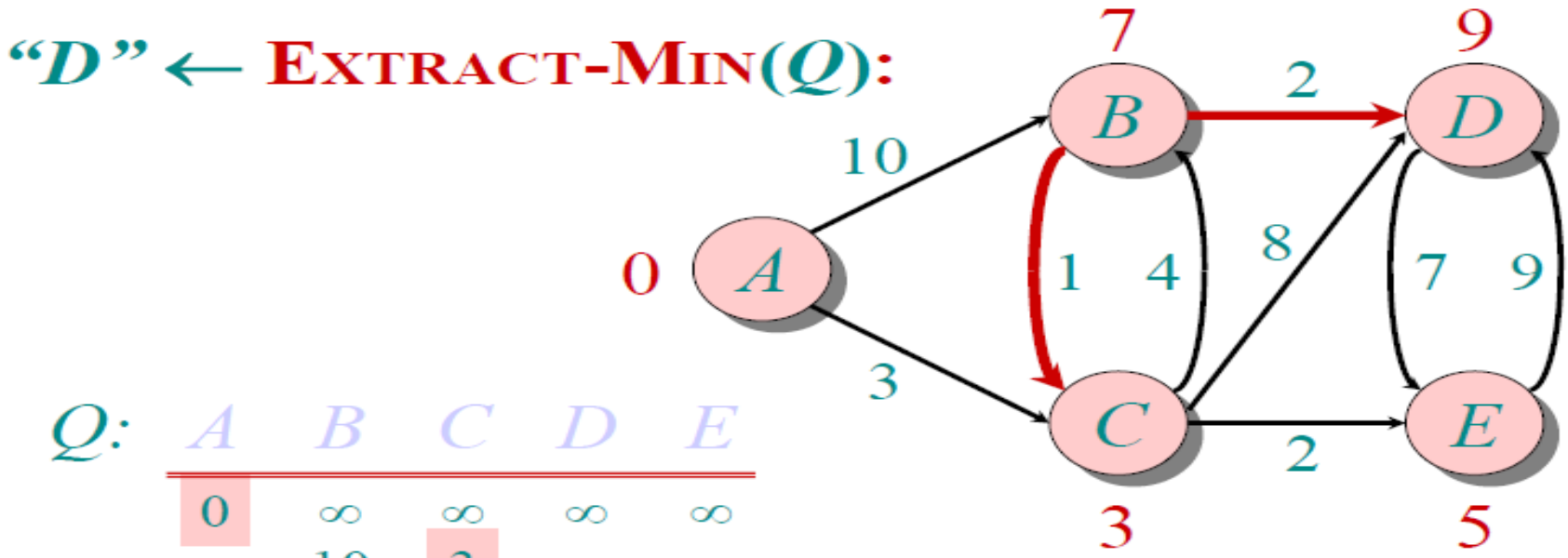
Relax all edges leaving *B*:



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



Correctness

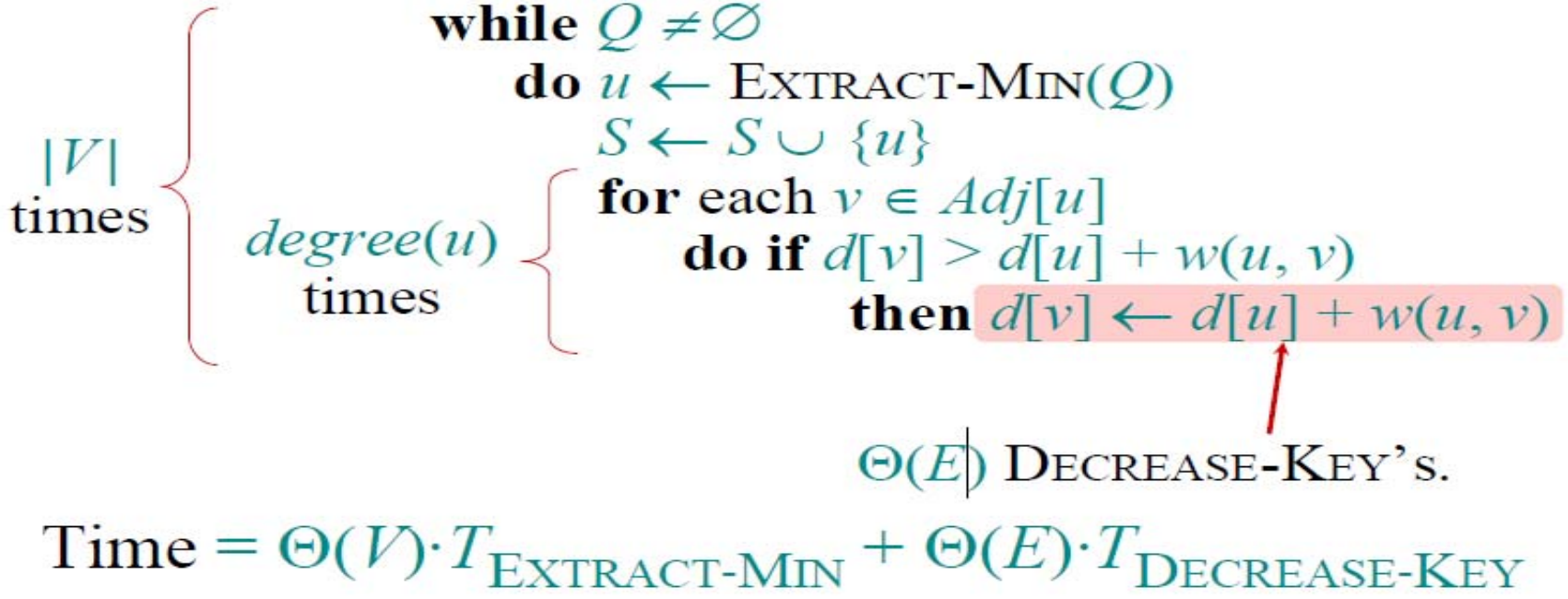
Lemma. Initializing $d[s] \leftarrow 0$ and $d[v] \leftarrow \infty$ for all $v \in V - \{s\}$ *establishes* $d[v] \geq \delta(s, v)$ for all $v \in V$, and this invariant is maintained over any sequence of relaxation steps.

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

Analysis of Dijkstra



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

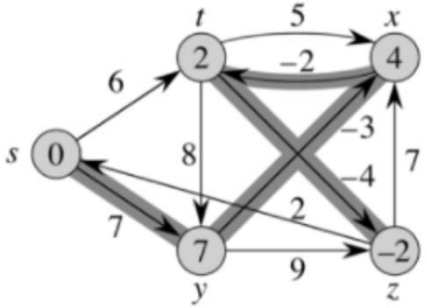
Analysis of Dijkstra (cont'd)

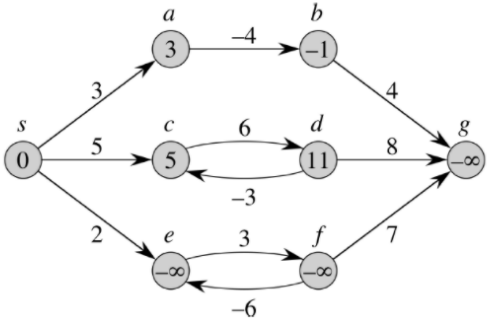
$$\text{Time} = \Theta(V)\cdot T_{\text{EXTRACT-MIN}} + \Theta(E)\cdot T_{\text{DECREASE-KEY}}$$

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

The Bellman-Ford Algorithm

- Dijkstra's algorithm doesn't work when there are negative edges:
  - Intuition – we can not be greedy any more on the assumption that the lengths of paths will only increase in the future
- Bellman-Ford algorithm solves the single-source shortest-paths problem in the general case in which edge weights may be negative.
  - detects negative cycles (returns *false*) or returns the shortest path-tree
  - Returns True if no negative-weight cycles reachable from s, False otherwise.





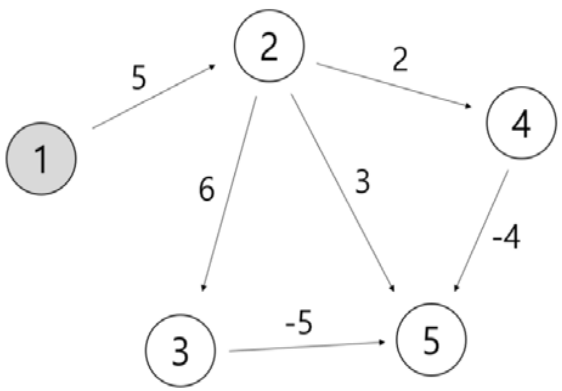
The Bellman-Ford Algorithm (cont'd)

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i ← 1 to |V[G]| − 1
3        do for each edge (u, v) ∈ E[G]
4              do RELAX(u, v, w)
5  for each edge (u, v) ∈ E[G]
6        do if d[v] > d[u] + w(u, v)
7              then return FALSE
8  return TRUE
```

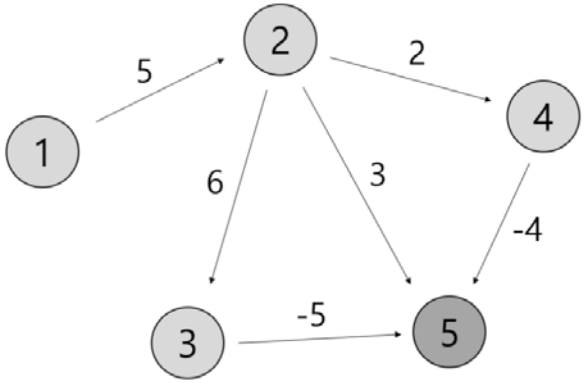
- ***Core:*** The first **for** loop relaxes all edges $|V|$ -1 times.
- ***Time:*** *$(|V|-1)|E| + |E| = O(VE)$*

The Bellman-Ford Algorithm (cont'd)

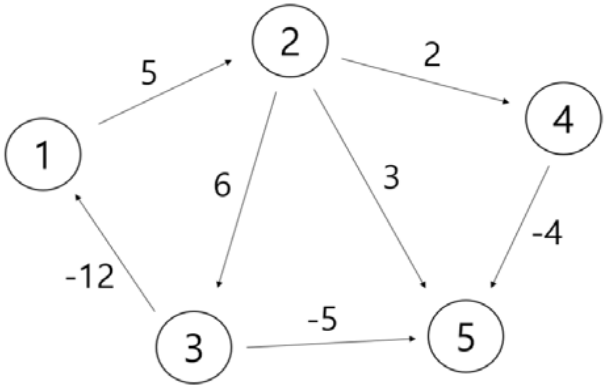
- Examples
  - Values you get on each pass and how quickly it converges depends on order of relaxation.
  - But guaranteed to converge after |*V*| -1 passes, assuming no negative-weight cycles.



(a-1)



(a-2)



(b)

The Bellman-Ford Algorithm (cont'd)

Proof Use path-relaxation property.

- Let v be reachable from s, and let $p = v_0, v_1, \ldots, v_k$ be a shortest path from s to v, where $v_0 = s$ and $v_k = v$. Since p is acyclic, it has $\leq |V| - 1$ edges, so $k \leq |V| - 1$.

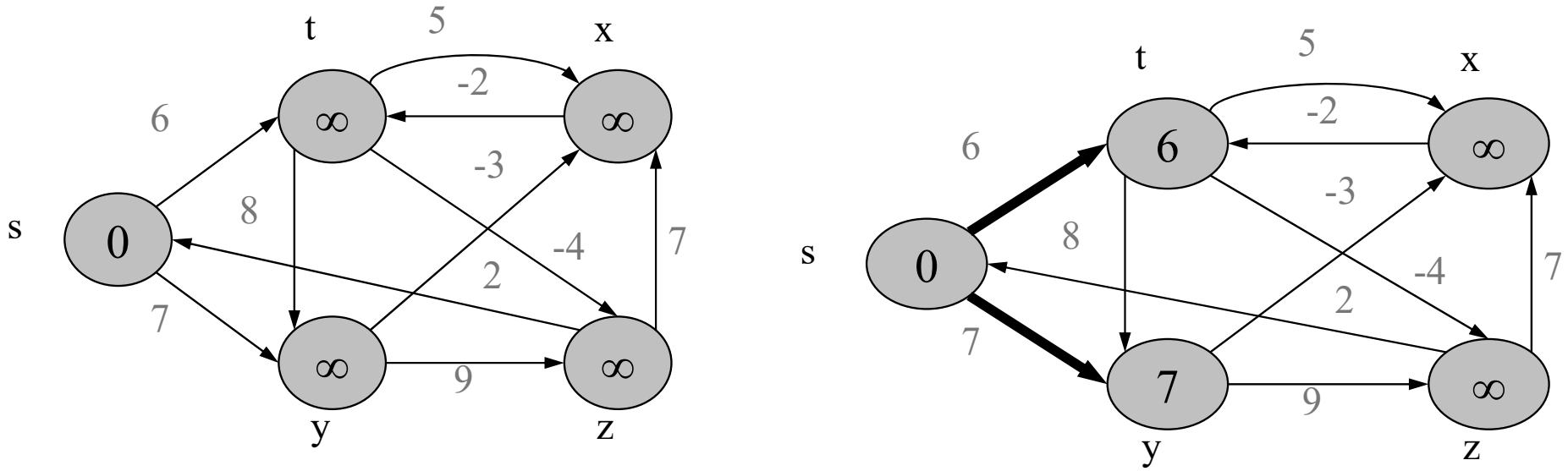
Each iteration of the for loop relaxes all edges:

- First iteration relaxes (v_0, v_1).
- Second iteration relaxes (v_1, v_2).
- K^{th} iteration relaxes (v_{k-1}, v_k).

By the path-relaxation property,

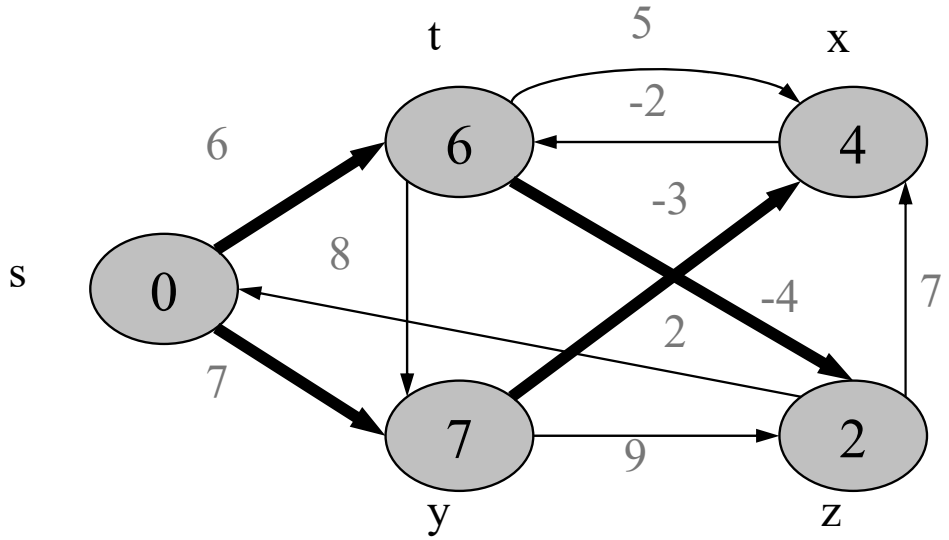
$d[v] = d[v_k] = \delta(s, v_k) = \delta(s, v)$.

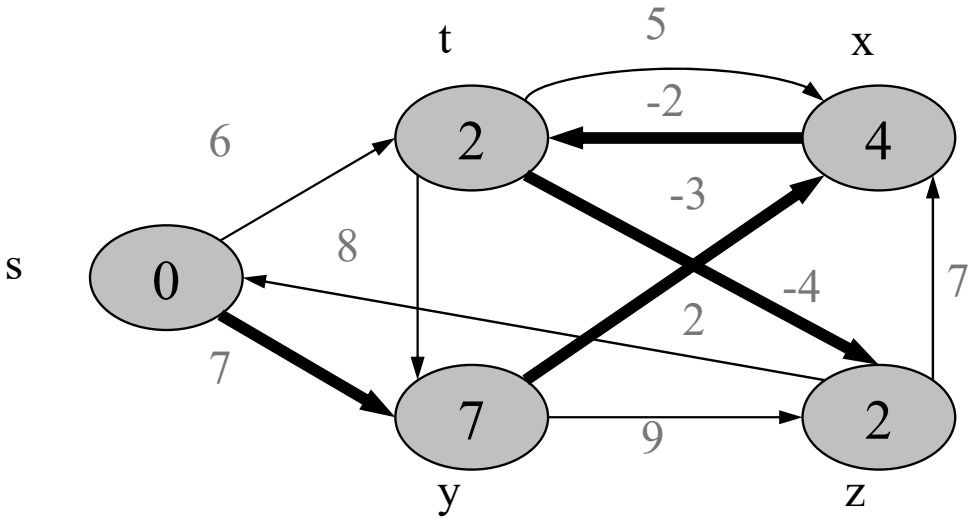
The Bellman-Ford Algorithm (cont'd)



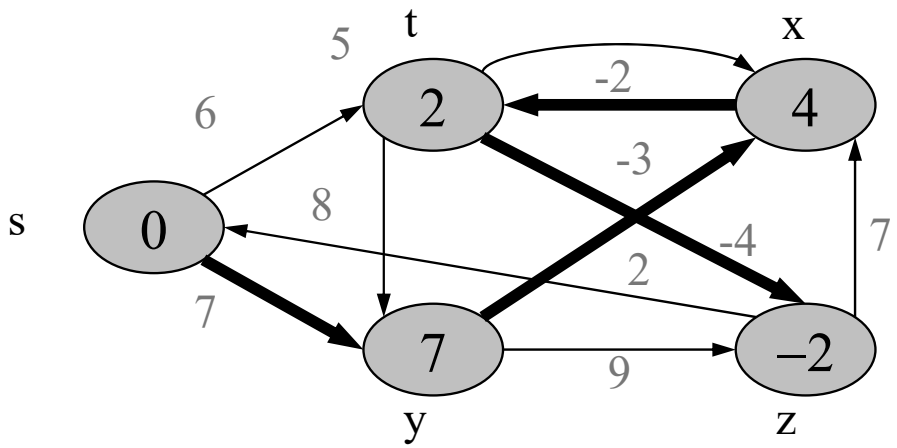
In this example, each path relaxes the edges in the order (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y).

The Bellman-Ford Algorithm (cont'd)





The Bellman-Ford Algorithm (cont'd)



- Bellman-Ford running time:
  - (|V|-1)|E| + |E| = O(VE)

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i ← 1 to |V[G]| − 1
3       do for each edge (u, v) ∈ E[G]
4            do RELAX(u, v, w)
5  for each edge (u, v) ∈ E[G]
6       do if d[v] > d[u] + w(u, v)
7            then return FALSE
8  return TRUE
```

Problem 10: Contest trip

- Tomorrow morning Tina must travel from Seoul to Daejeon to compete in the regional programming contest.
  - Since she is afraid of arriving late and being excluded from the contest, she is looking for the train which gets her to Daejeon as early as possible.
  - However, she dislikes getting to the station too early, so if there are several schedules with the same arrival time then she will choose the one with the latest departure time.
- Tina asks you to help her with her problem. You are given a set of railroad schedules from which you must compute the train with the earliest arrival time and the fastest connection from one location to another.
  - Fortunately, is very experienced in changing trains and can do this instantaneously, i.e., in zero time!

Problem 10:Contest trip (Cont'd)

- Input
  - Each schedule consists of three parts. The first part lists the names of all cities connected by the railroads. It starts with a number $1 < C \leq 100$, followed by C lines containing city names. All names consist only of letters.
  - The second part describes all the trains running during a day. It starts with a number $T \leq 100$ followed by T train descriptions. Each of them consists of one line with a number $t_i \leq 100$ and then t_i more lines, each with a time and a city name, meaning that passengers can get on or off the train at that time at that city.
  - The final part consists of three lines: the first containing the earliest possible starting time, the second the name of the city where she starts, and the third with the destination city. The start and destination cities are always different.

Problem 10:Contest trip (Cont'd)

- Output
  - If a connection exists, print the two lines containing zero padded timestamps and locations as shown in the example. Use blanks to achieve the indentation. If no connection exists on the same day (i.e., arrival before midnight), print a line containing "No connection".

Problem 10:Contest trip (Cont'd)

- Sample input
  - 1
  - 3
  - Seoul
  - Suwon
  - Daejeon
  - 3
  - 2
  - 0949 Seoul
  - 1006 Suwon
  - 2
  - 1325 Seoul
  - 1550 Daejeon
  - 2
  - 1205 Suwon
  - 1411 Daejeon
  - 0800
  - Seoul
  - Daejeon

Problem 10: Contest trip (Cont'd)

- Sample output
  - ✓ Departure 0949 Seoul
  - ✓ Arrival 1411 Daejeon

THANK YOU