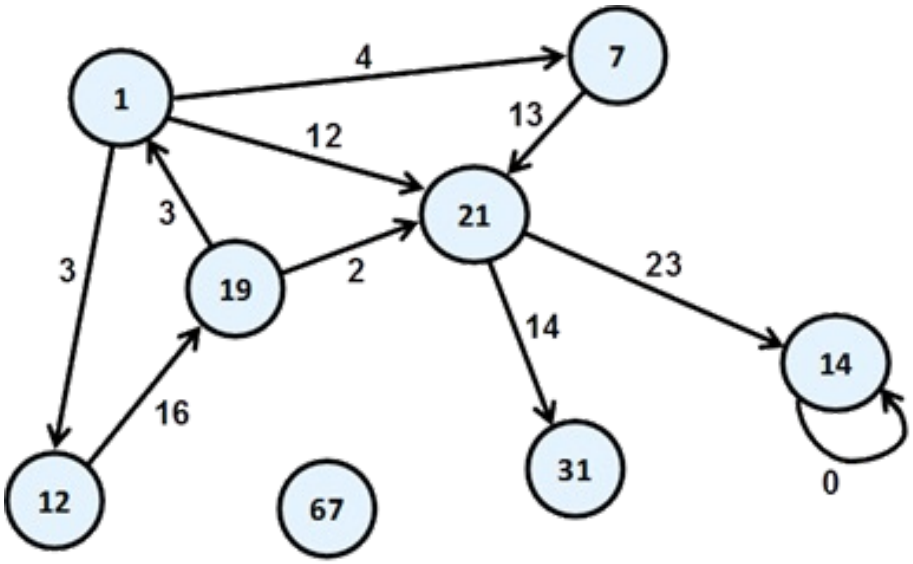
Graph Algorithms 2



NYIT CSCI-651

Shortest Path Algorithms

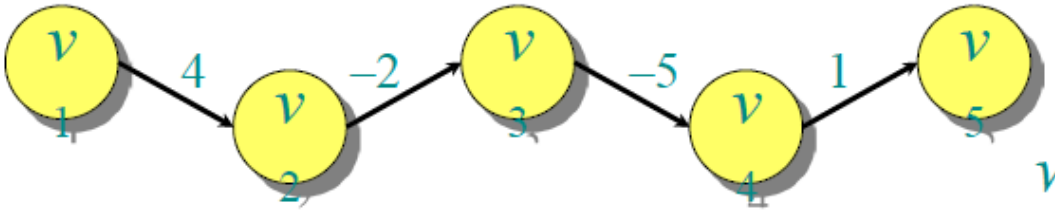
Weight of a paths

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

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

The weight w(p) of path p = <v_0 , v_1 , . . . , v_k > is the sum of the weights of its constituent edges

Example:

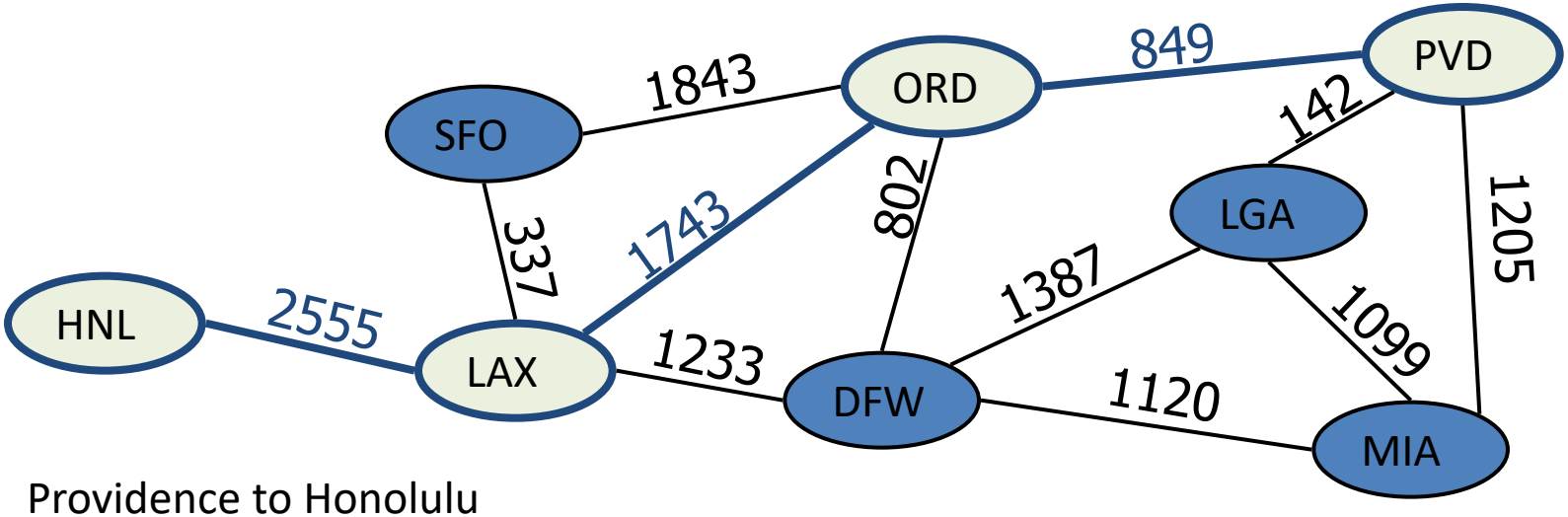


$w(p) = -2$

Shortest Paths

- Given a weighted graph and two vertices u and v, we want to find a path of minimum total weight between u and v.
  - Length of a path is the sum of the weights of its edges.

- Some applications
  - Internet packet routing
  - Flight reservations
  - Driving directions
  - …



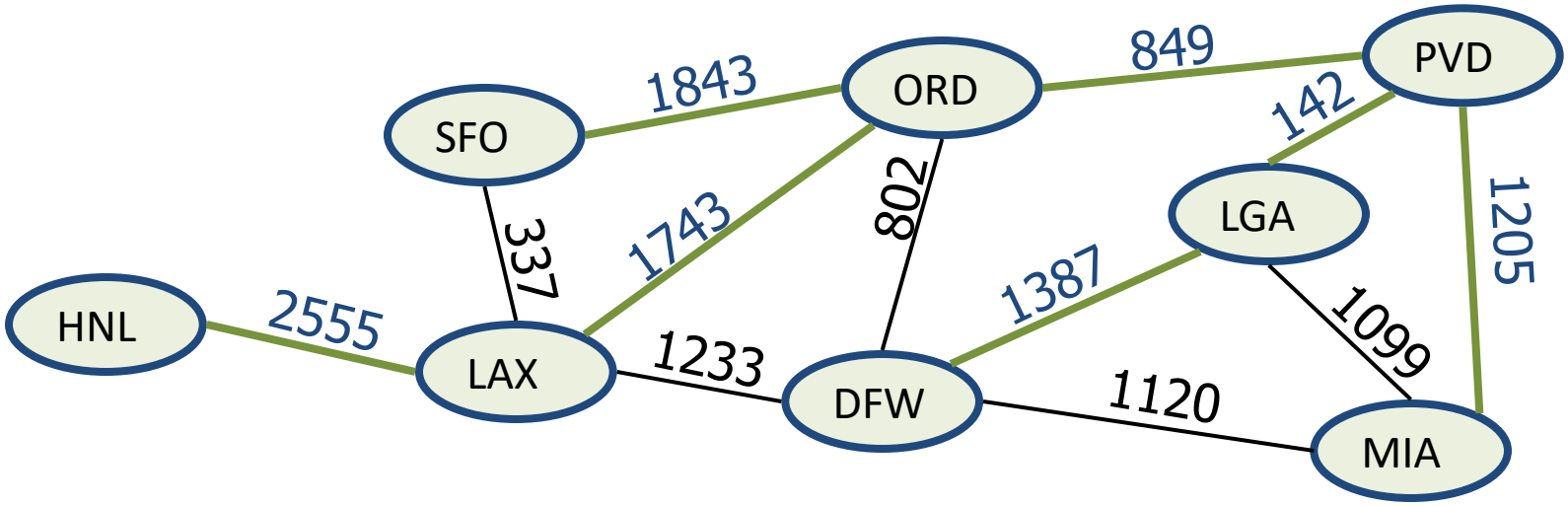
Providence to Honolulu

Shortest Path Properties

Property 1: A sub-path of a shortest path is itself a shortest path

Property 2: There is a tree of shortest paths from a start vertex to all the other vertices

Example: Tree of shortest paths from Providence



Basic Categories

- Single source vs. all-pairs
  - Single Source Shortest Path: SSSP
  - All-pairs Shortest Path: APSP
- Weighted vs. unweighted
  - Can edges be negative?
  - Can there be negative cycles?

- SSSP on Unweighted Graph
  - Just use BFS!

Types of Shortest Path Problems

- SSSP: single source shortest path
  - find the shortest path from a given vertex to all the other nodes
  - for undirected, unweighted graphs use **BFS**
  - for directed, weighted graph use:
    - **Dijkstra, Bellman-Ford**

- APSP: all pairs shortest path
  - find the shortest path between all pairs of nodes in the graphs
    - **Floyd-Warshall**

Optimal substructure of a shortest path

- Shortest-paths algorithms typically rely on the property that a shortest path between two vertices contains other shortest paths within it.

Lemma 24.1 (Subpaths of shortest paths are shortest paths)
Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$, let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path from vertex v_0 to vertex v_k and, for any i and j such that $0 \le i \le j \le k$, let $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ be the subpath of p from vertex v_i to vertex v_j. Then, p_{ij} is a shortest path from v_i to v_j.

Single Source Shortest Path

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

- Build a shortest path tree (SPT) from a source node
- No negative edge weights allowed.

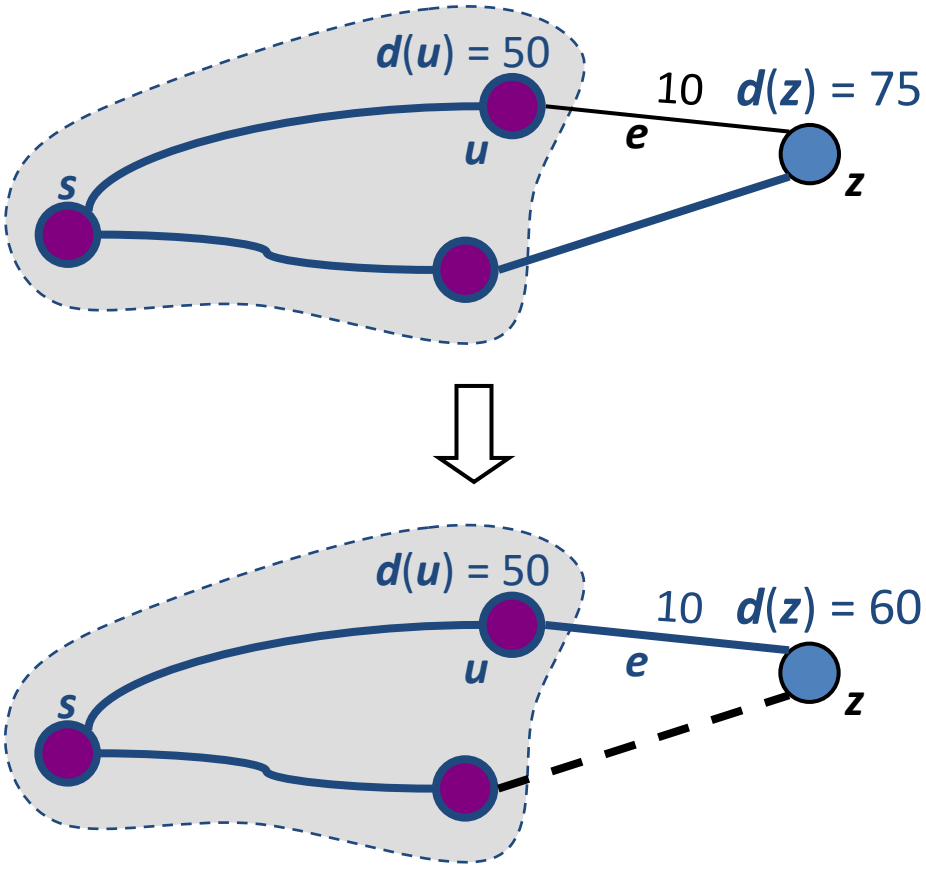
- Similar to Prim's algorithm (Greedy)
  - Grow a tree gradually, advancing from vertices taken from a priority queue of "distances"
  - Instead of a MST, we are building a SPT

Dijkstra's Algorithm

- It computes the shortest distance of all the vertices from a given source vertex ***s***
- It grows a cloud of vertices, beginning with ***s*** and eventually covering all the vertices. It store with each vertex ***v*** a value ***d***(***v***) representing the distance of ***v*** from ***s*** in the subgraph consisting of the cloud and its adjacent vertices.
- At each step
  - it adds to the cloud a vertex ***u*** outside the cloud with the shortest ***d***(***u***)
  - it updates the labels of the vertices adjacent to ***u***

Edge Relaxation

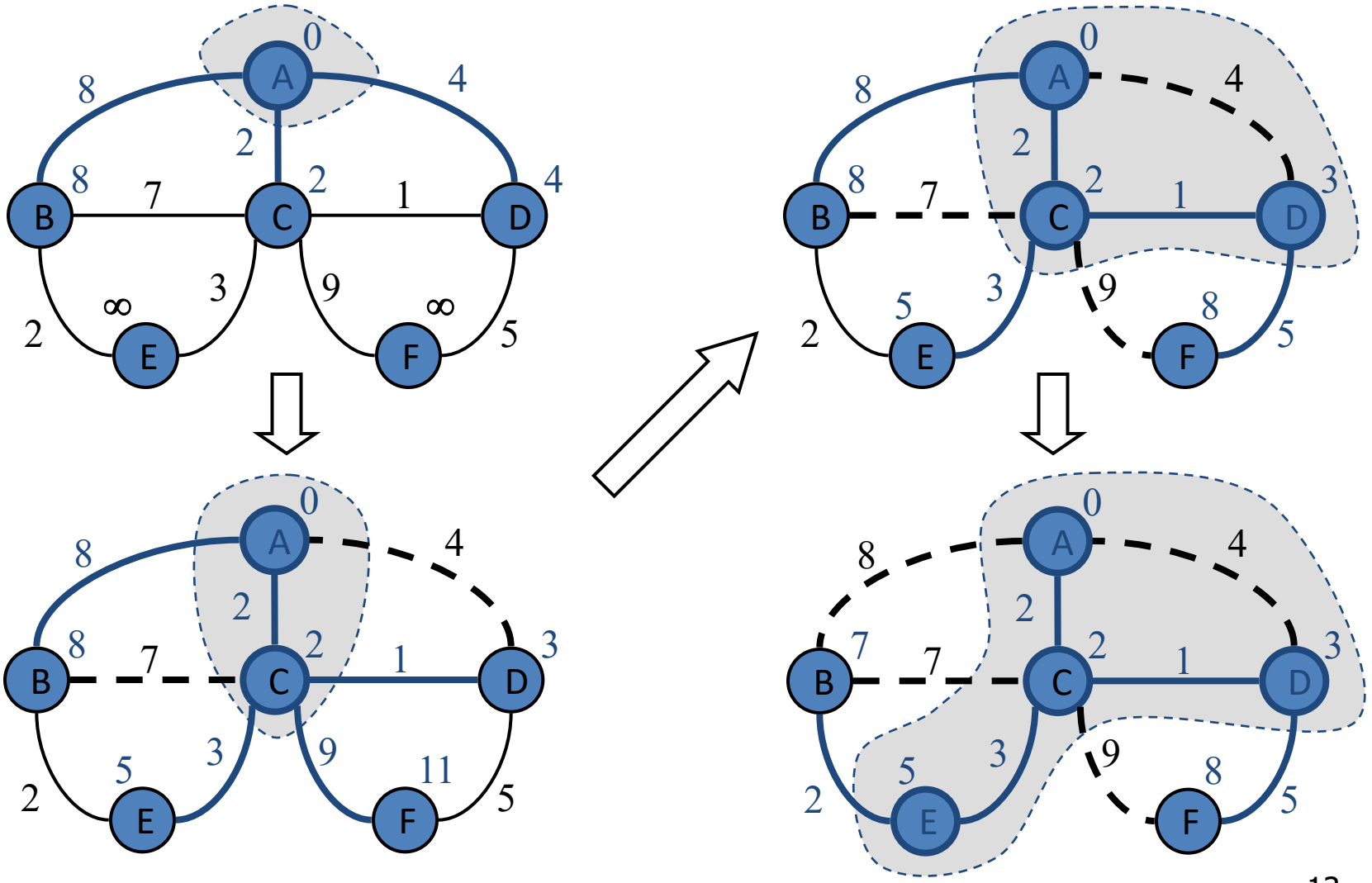
- Consider ***e*** **=** (***u,z***) such that
  - ***u*** is the vertex most recently added to the cloud
  - ***z*** is not in the cloud



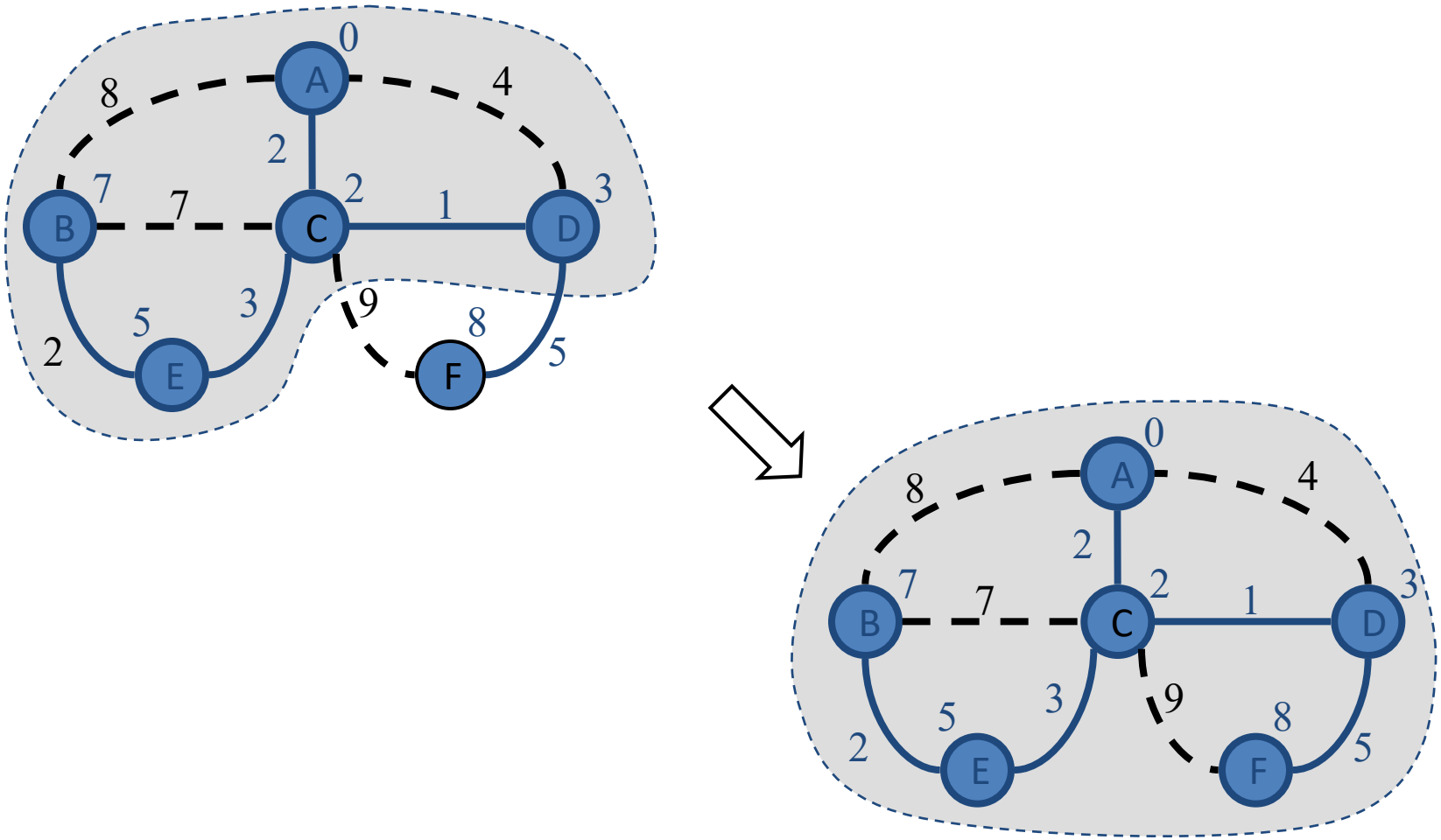
- The relaxation of ***e*** updates distance ***d***(***z***) as follows:

  d(***z***) ← min{***d***(***z***), ***d***(***u***) + ***weight***(***e***)}

Example



Example (cont.)



Dijkstra's Algorithm

Algorithm ShortestPath(G, s):

Input: A weighted graph G with nonnegative edge weights, and a distinguished vertex s of G.

Output: The length of a shortest path from s to v for each vertex v of G.

Initialize $D[s] = 0$ and $D[v] = \infty$ for each vertex $v \neq s$.
Let a priority queue Q contain all the vertices of G using the D labels as keys.
while Q is not empty **do**
  {pull a new vertex u into the cloud}
  u = value returned by Q.remove_min()
  for each vertex v adjacent to u such that v is in Q **do**
    {perform the ***relaxation*** procedure on edge (u, v)}
    if $D[u] + w(u, v) < D[v]$ **then**
      $D[v] = D[u] + w(u, v)$
      Change to $D[v]$ the key of vertex v in Q.
return the label $D[v]$ of each vertex v

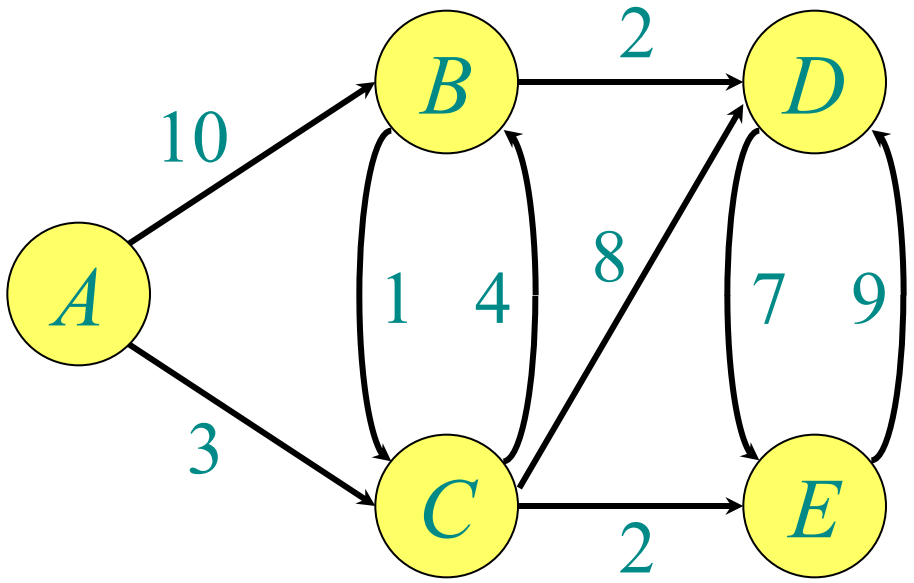
Dijkstra's Algorithm

$d[s] \leftarrow 0$
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

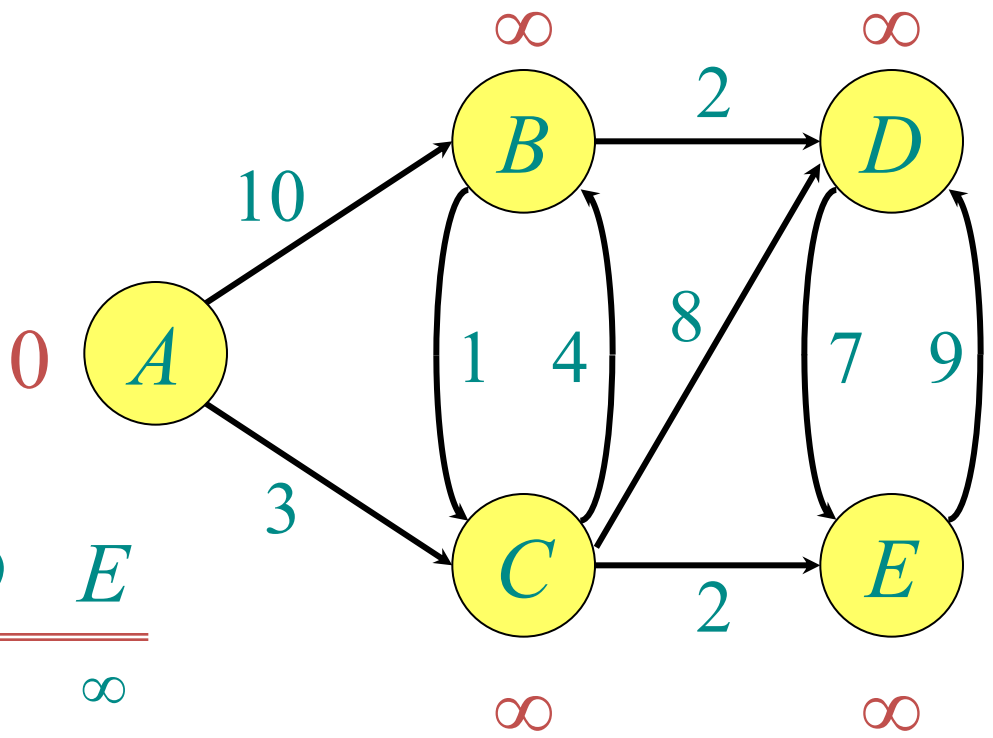
Demo of Dijkstra's Algorithm

Graph with nonnegative edge lengths:



Demo of Dijkstra's Algorithm

Initialize:

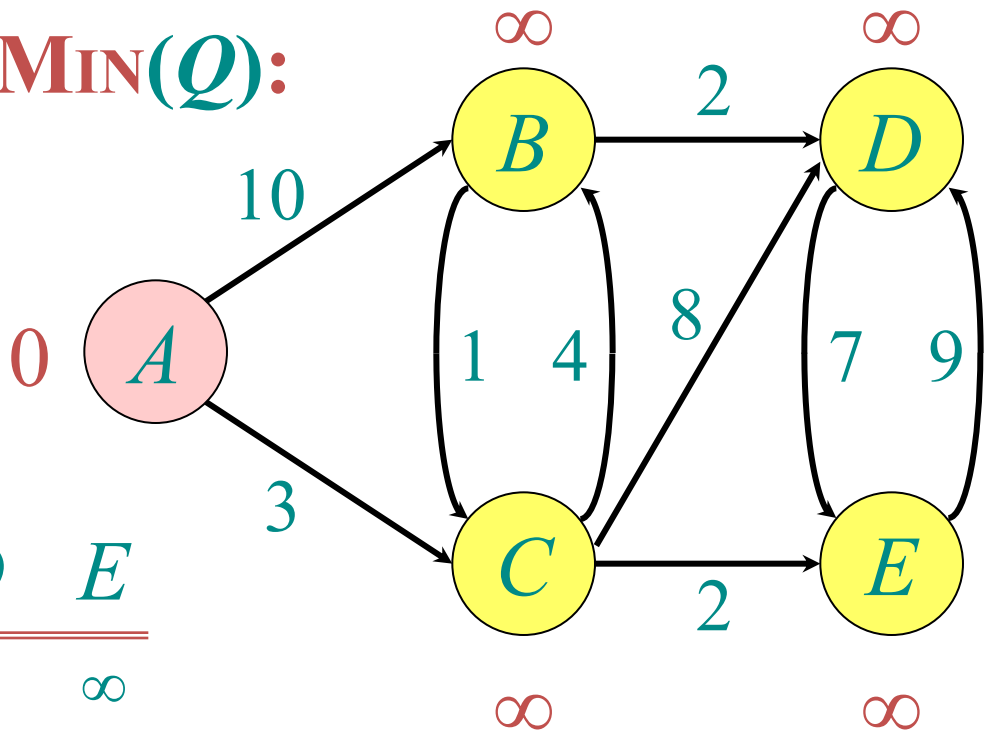


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: {}

Demo of Dijkstra's Algorithm

"A" ← EXTRACT-MIN(Q):

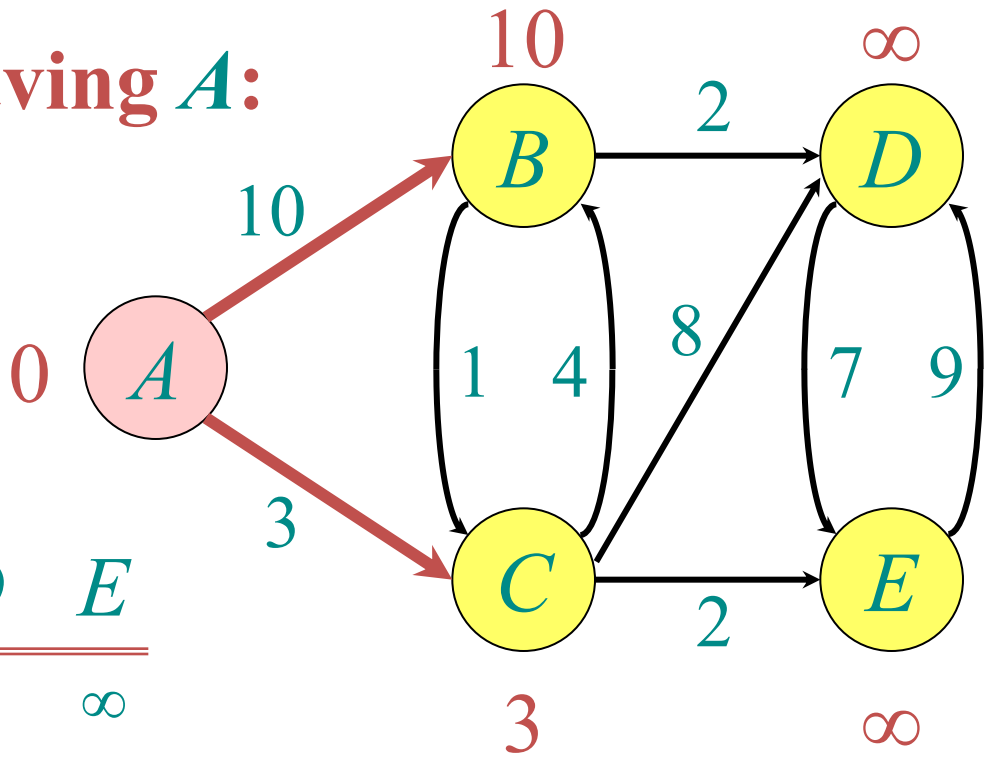


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: { A }

Demo of Dijkstra's Algorithm

Explore edges leaving *A*:

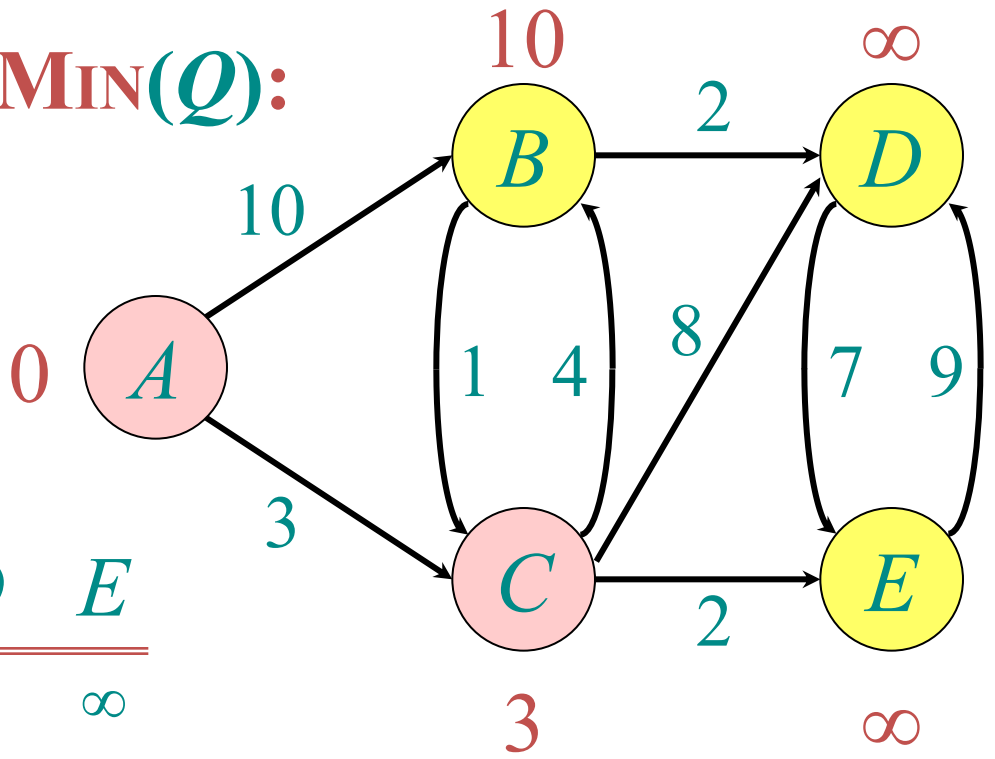


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { *A* }

Demo of Dijkstra's Algorithm

"C" ← EXTRACT-MIN(Q):

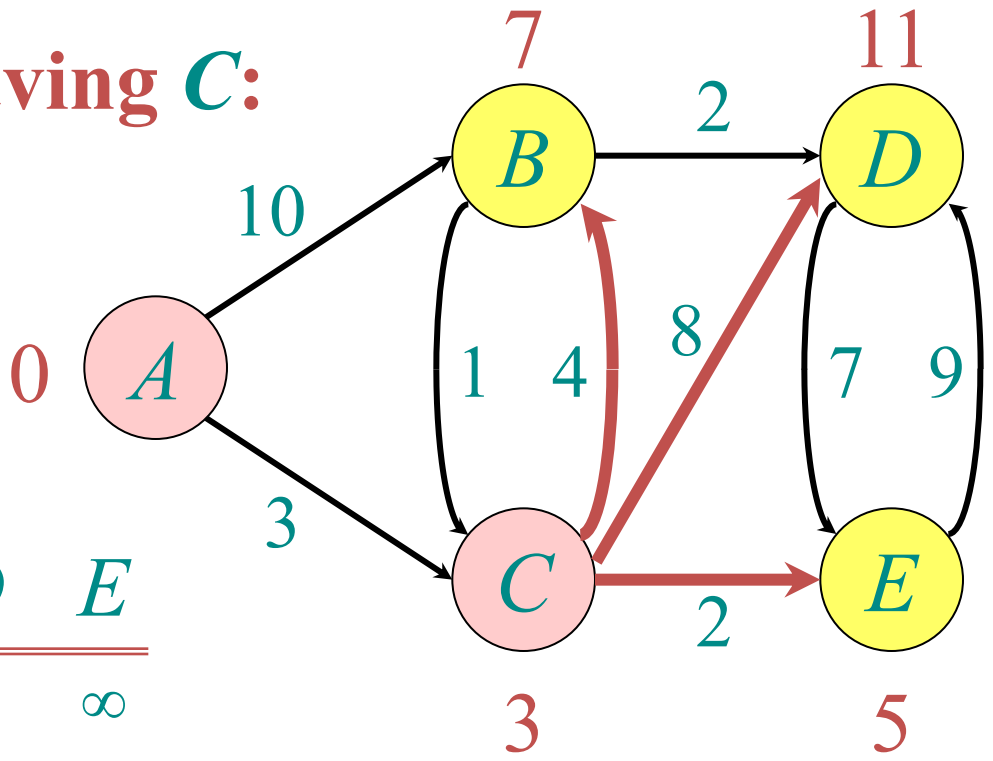


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A, C }

Demo of Dijkstra's Algorithm

Explore edges leaving *C*:

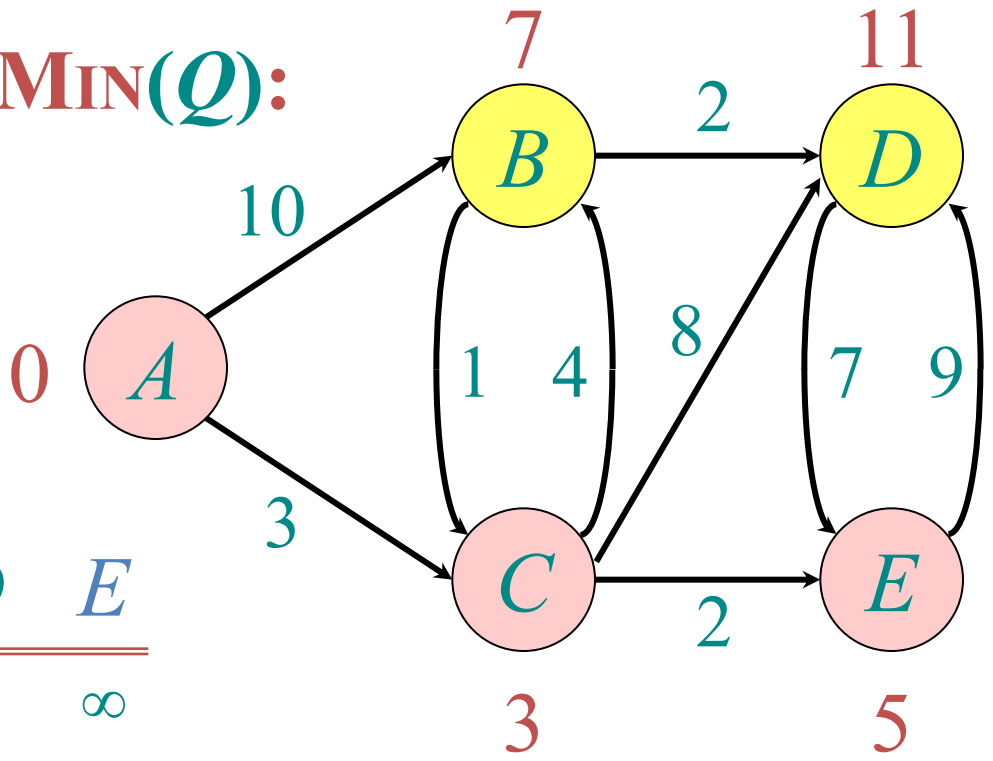


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { *A*, *C* }

Demo of Dijkstra's Algorithm

"E" ← EXTRACT-MIN(*Q*):

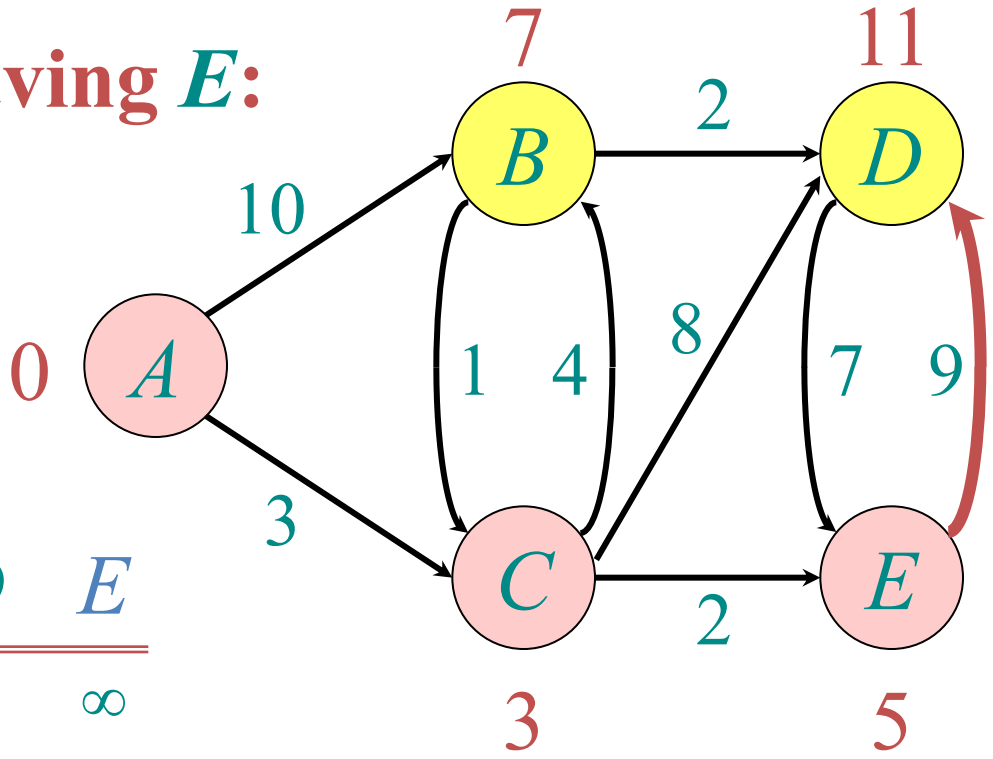


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

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

Demo of Dijkstra's Algorithm

Explore edges leaving *E*:

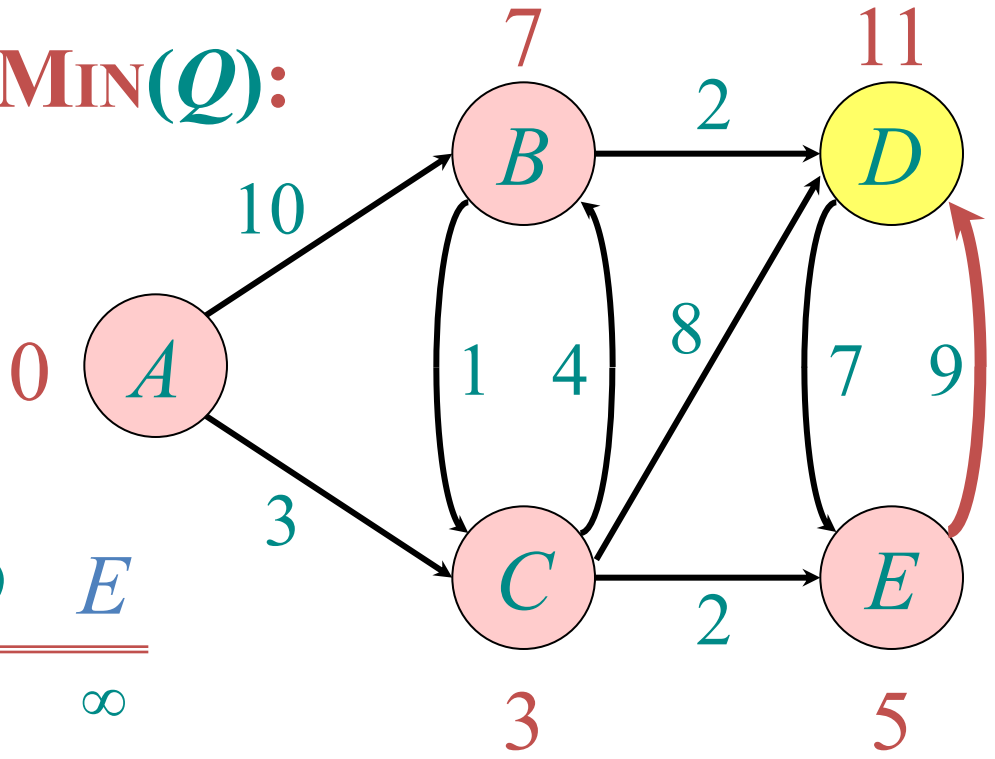


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

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

Demo of Dijkstra's Algorithm

"*B*" ← EXTRACT-MIN(*Q*):

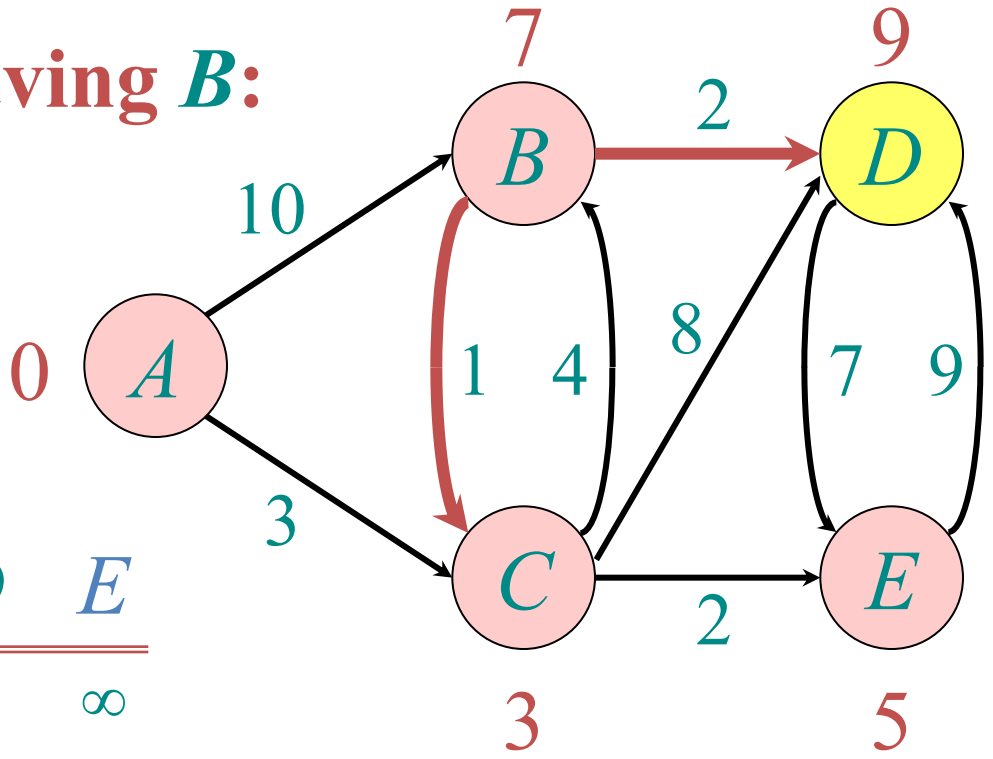


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

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

Demo of Dijkstra's Algorithm

Explore edges leaving *B*:

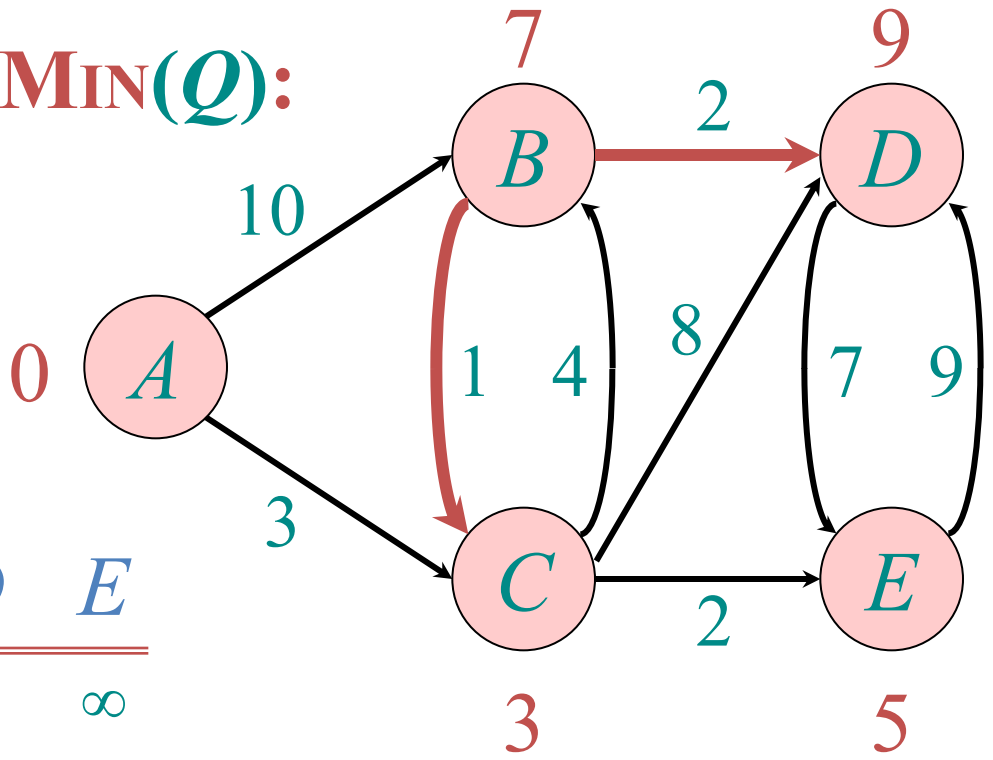


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

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

Demo 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* }

Dijkstra's Algorithm

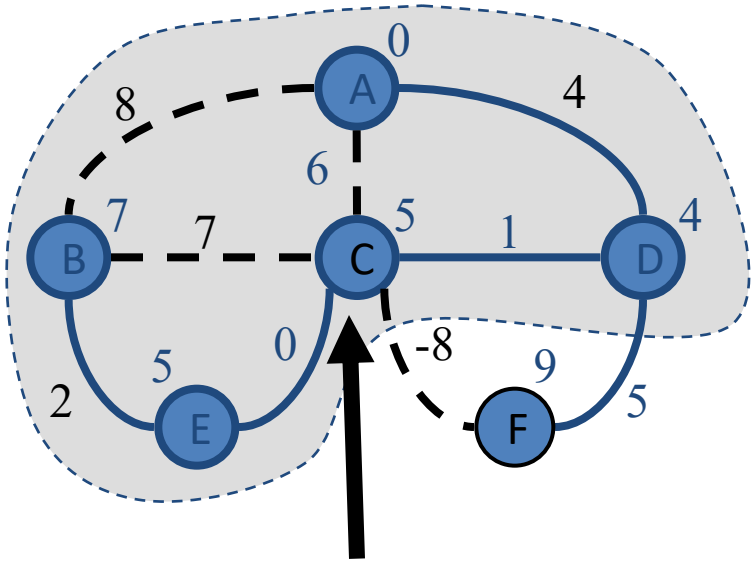
- The Dijkstra's Algorithm always chooses the "lightest" or "closest" vertex in V-S to add to set S, so it uses a **Greedy Strategy**.

- Complexity: $\theta(|E| + |V|Log|V|)$

Problem with negative-weight edges

- Dijkstra's algorithm is based on the greedy method. It adds vertices by increasing distance.

  – If a node with a negative incident edge were to be added late to the cloud, it could mess up distances for vertices already in the cloud.

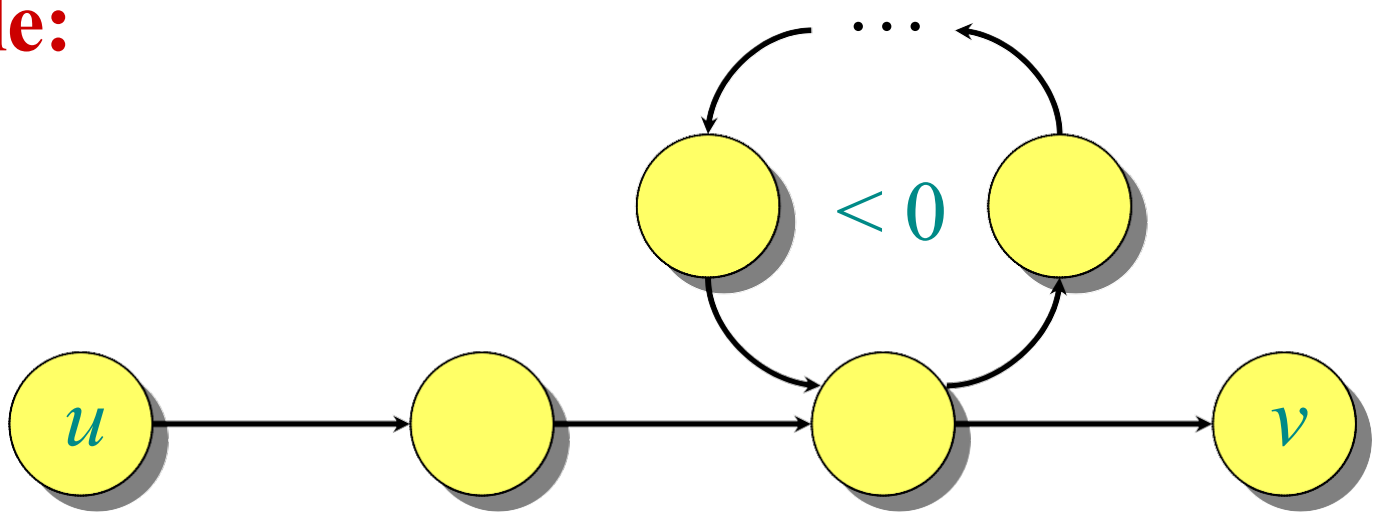


C's distance is 1, but it is in the cloud with d(C)=5!

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 lengths from a ***source*** $s \in V$ to all $v \in V$ or determines that a negative-weight cycle exists.

Bellman-Ford algorithm

Method: Relax the whole set of edges |V|-1 times.

$d[s] \leftarrow 0$
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 $d[v] \leftarrow d[u] + w(u, v)$

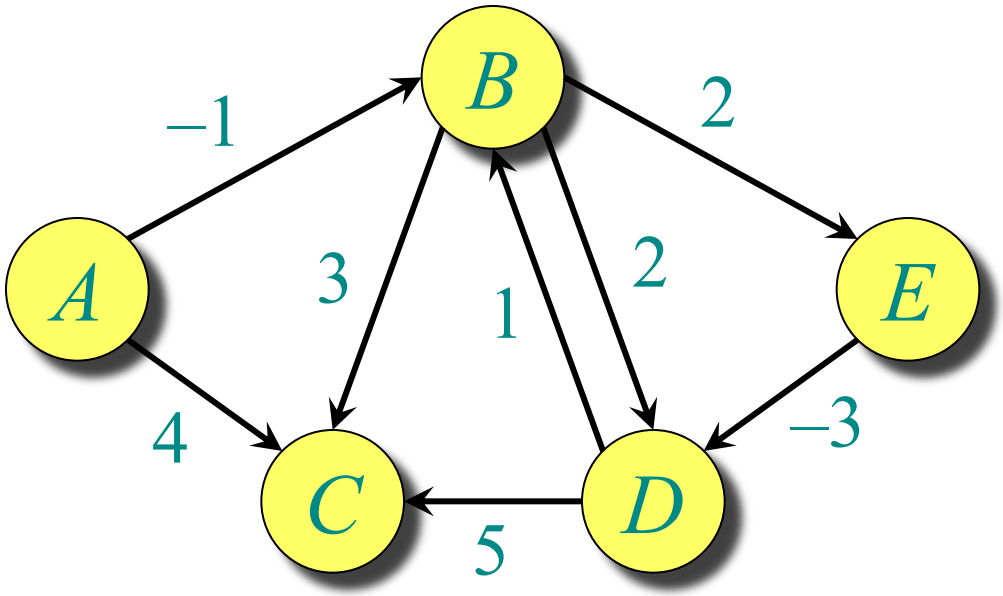
} ***relaxation step***

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, $d[v] = \delta(s, v)$, if no negative-weight cycles.

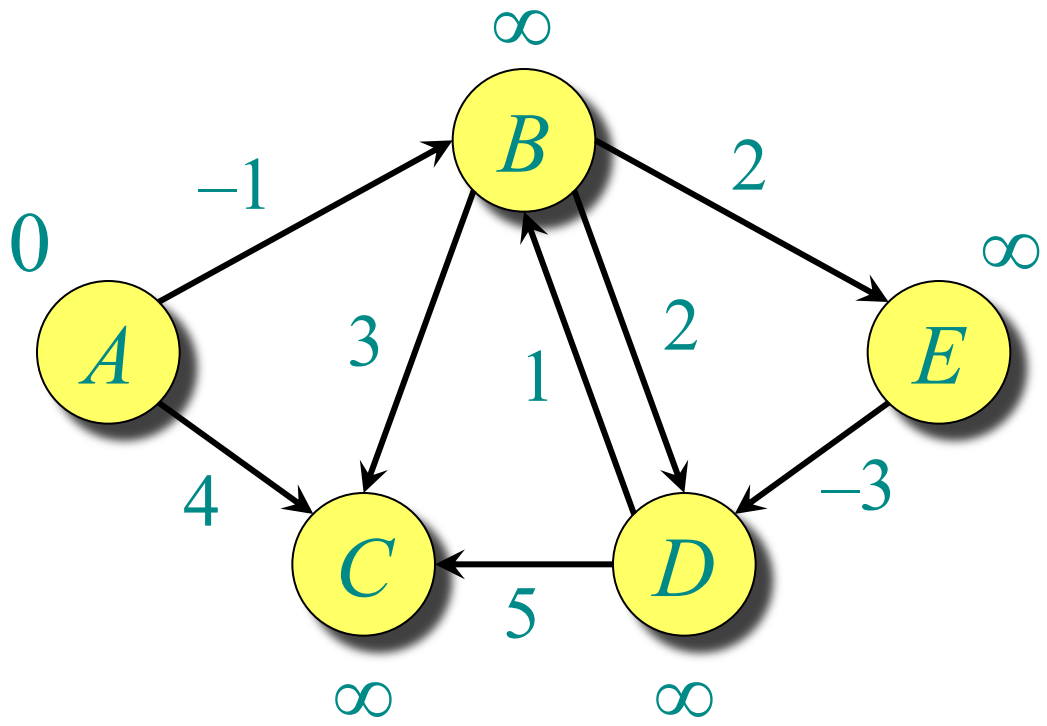
Time = $O(VE)$.

Example of Bellman-Ford



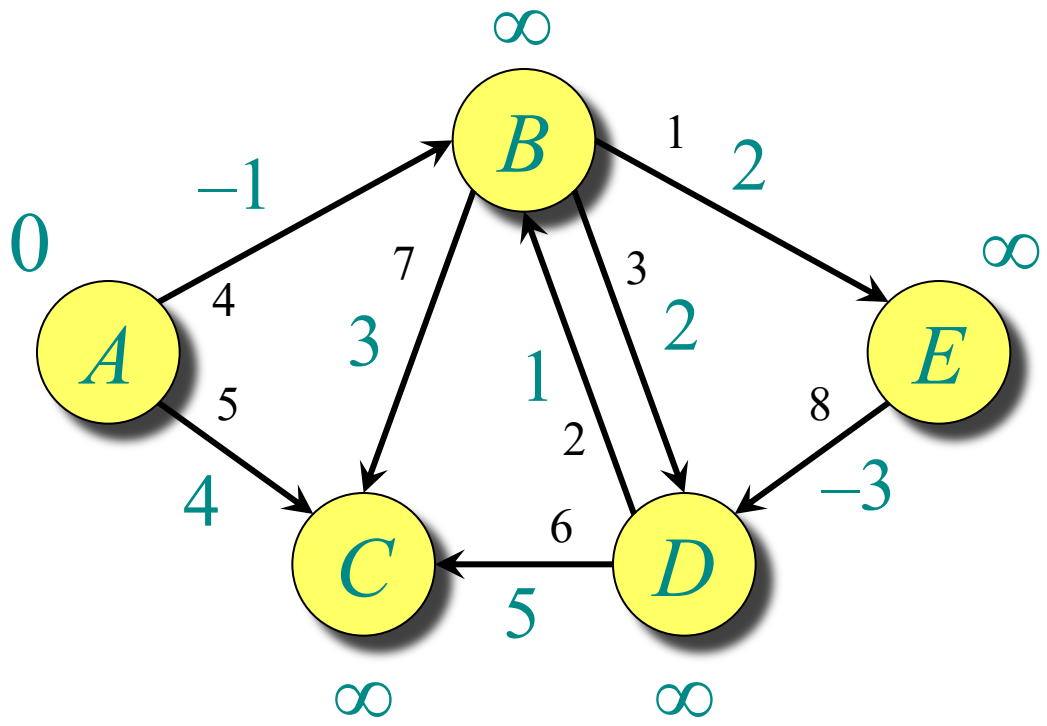
The demonstration is for a slightly different version of the algorithm that computes distances from the source node rather than distances to the destination node.

Example of Bellman-Ford



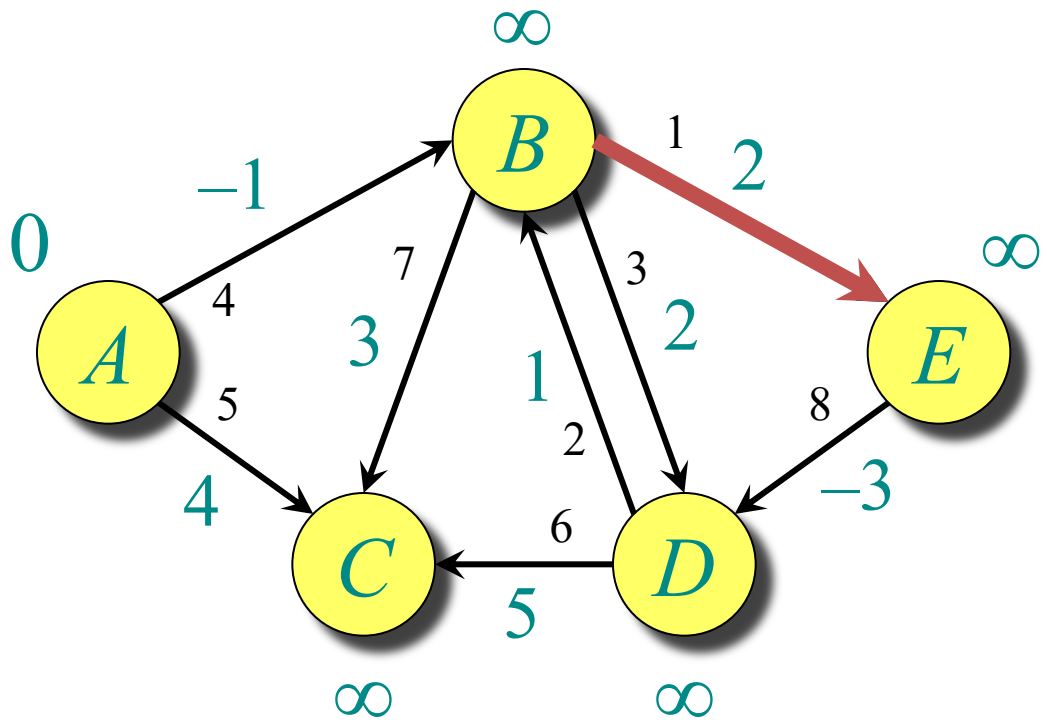
Initialization.

Example of Bellman-Ford

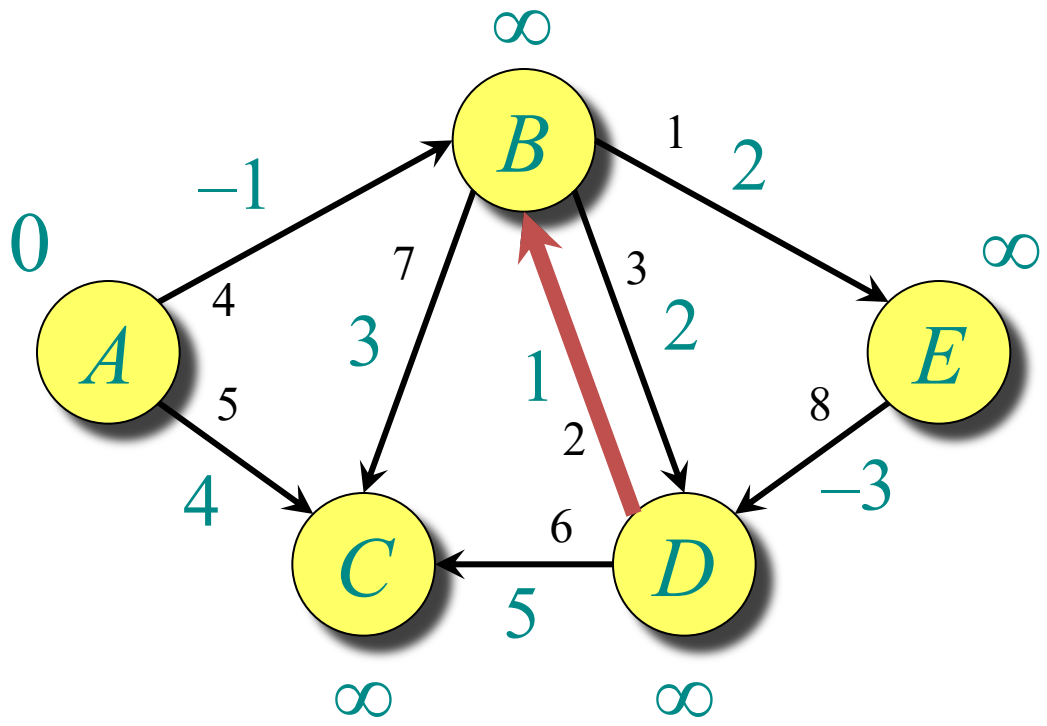


Order of edge relaxation.

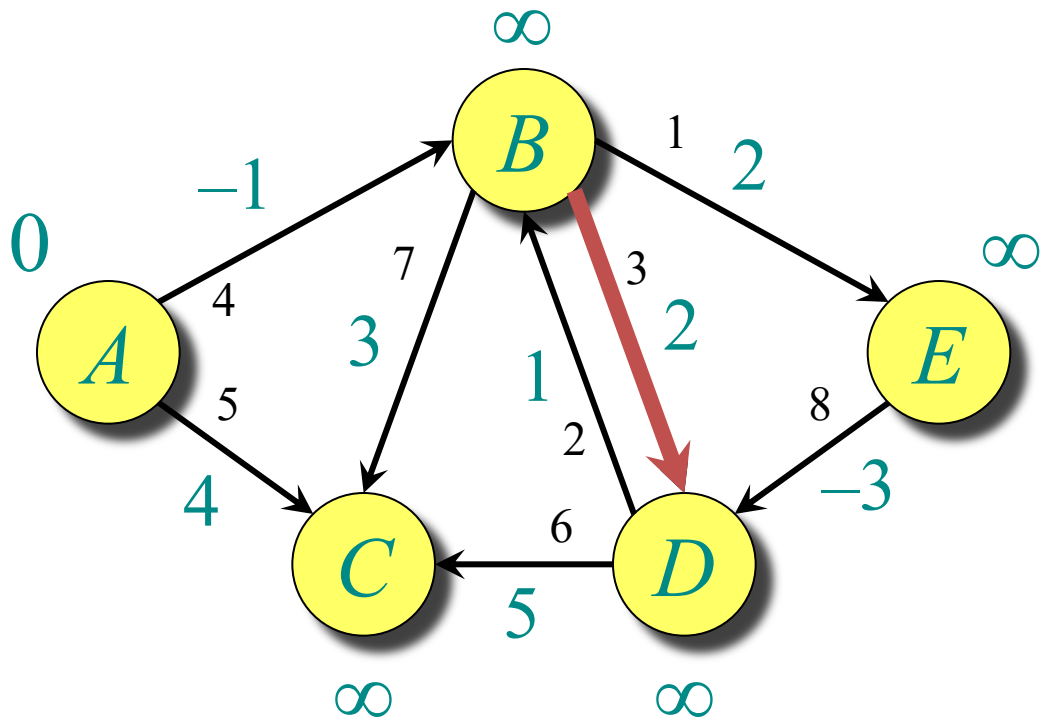
Example of Bellman-Ford



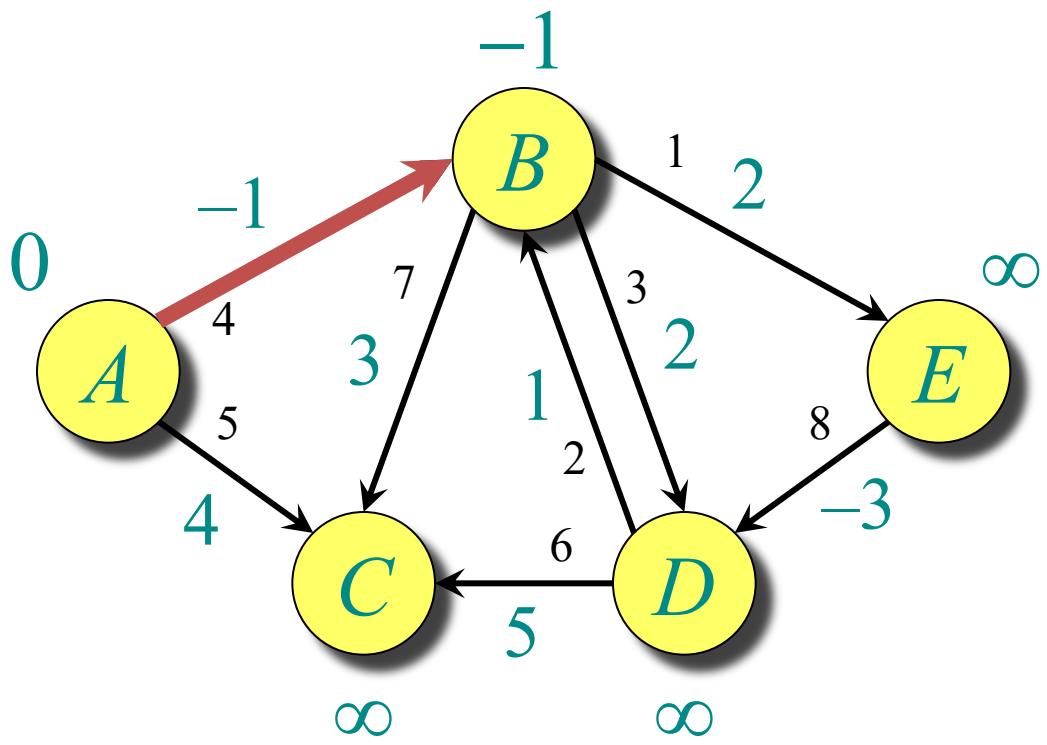
Example of Bellman-Ford



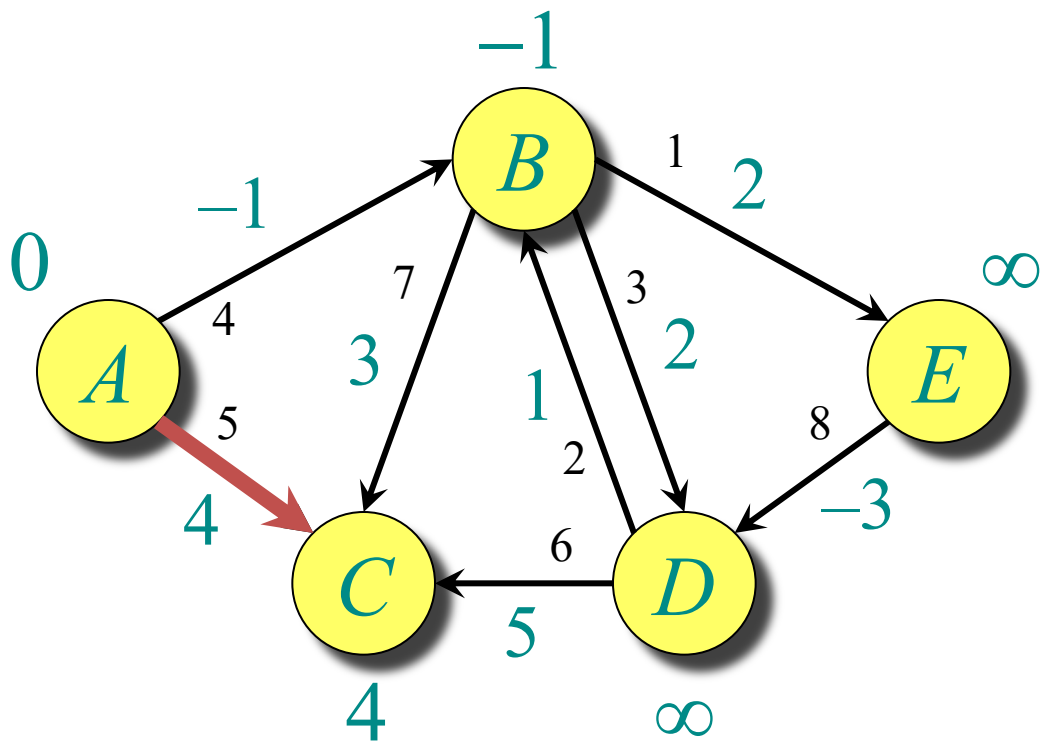
Example of Bellman-Ford



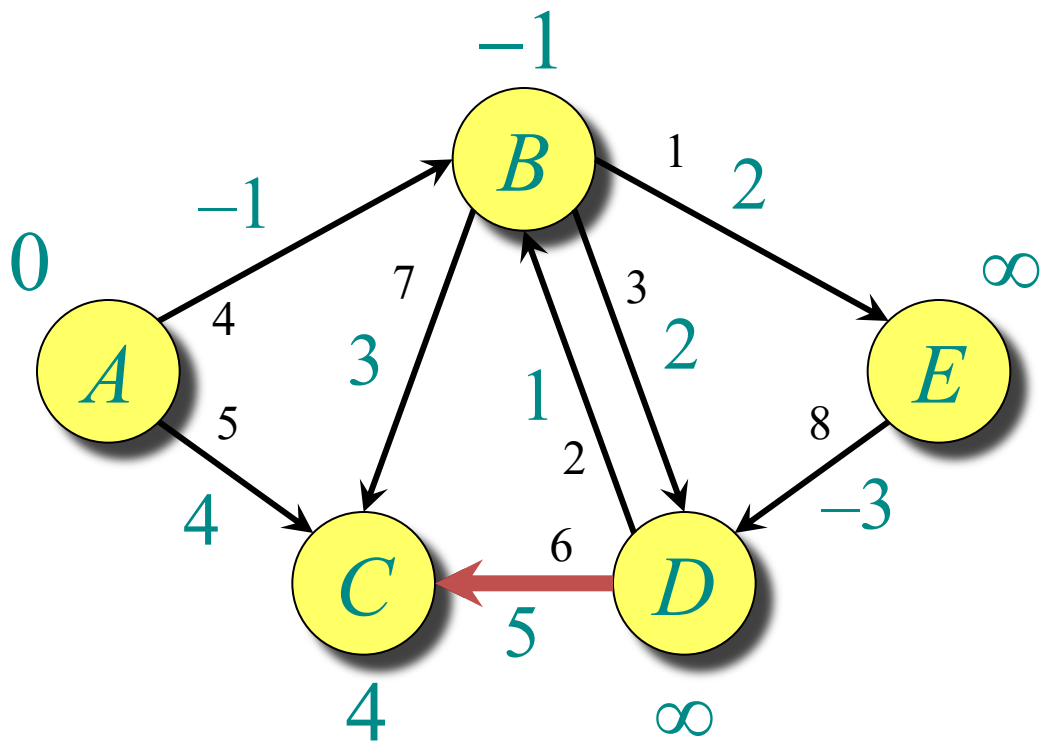
Example of Bellman-Ford



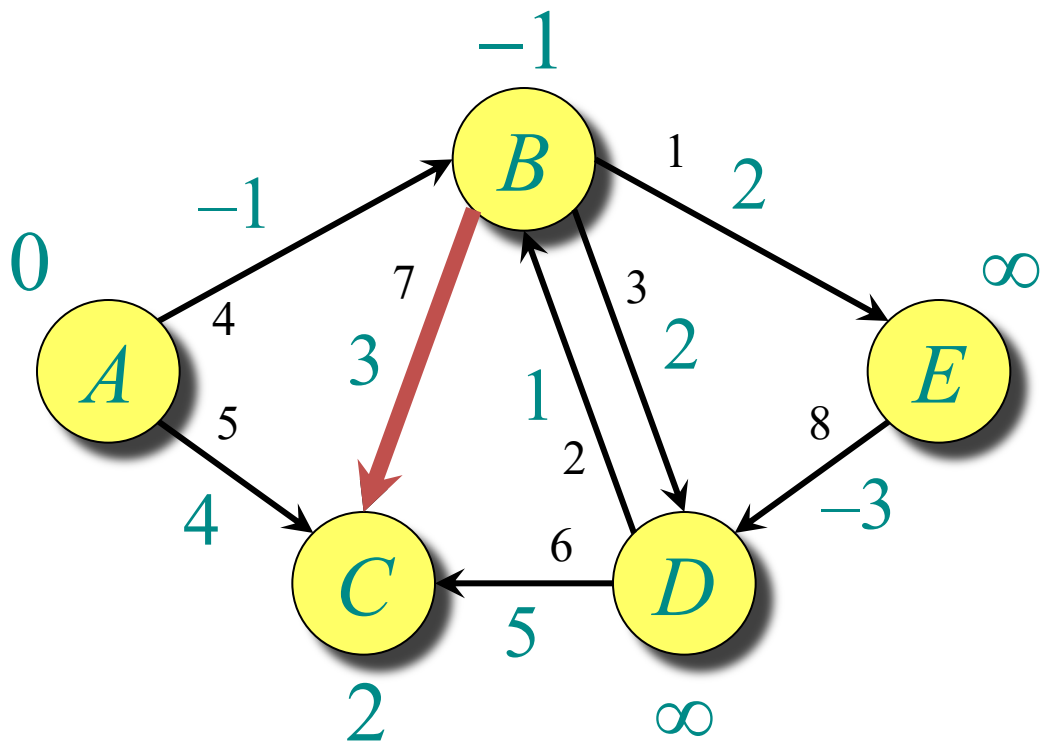
Example of Bellman-Ford



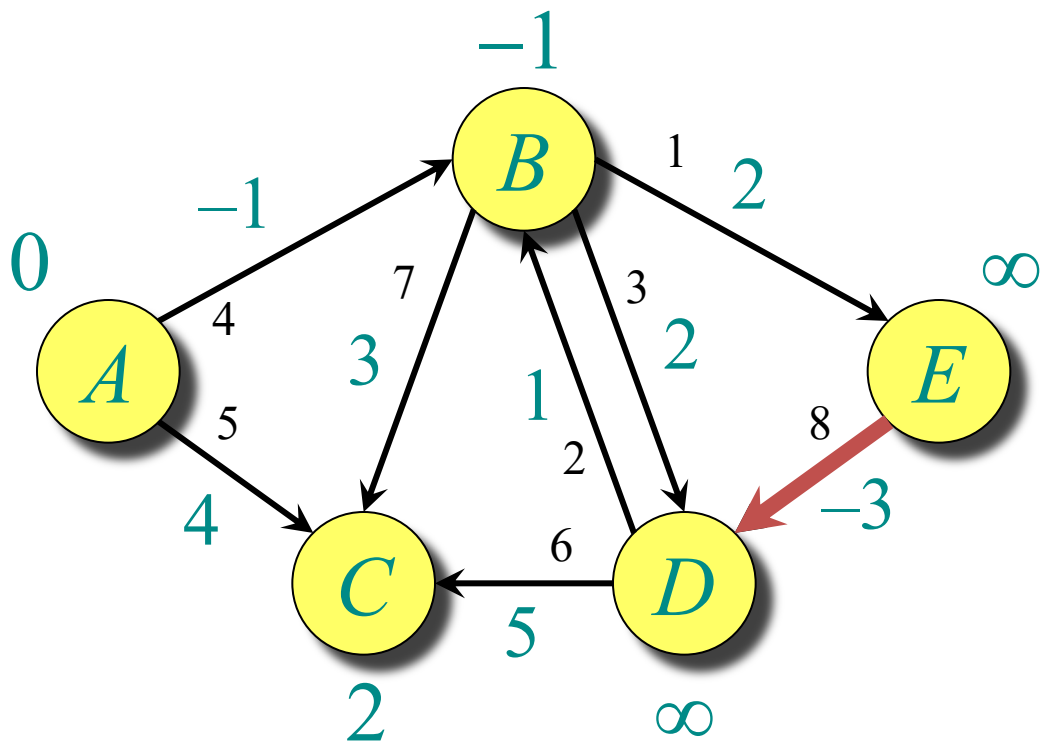
Example of Bellman-Ford



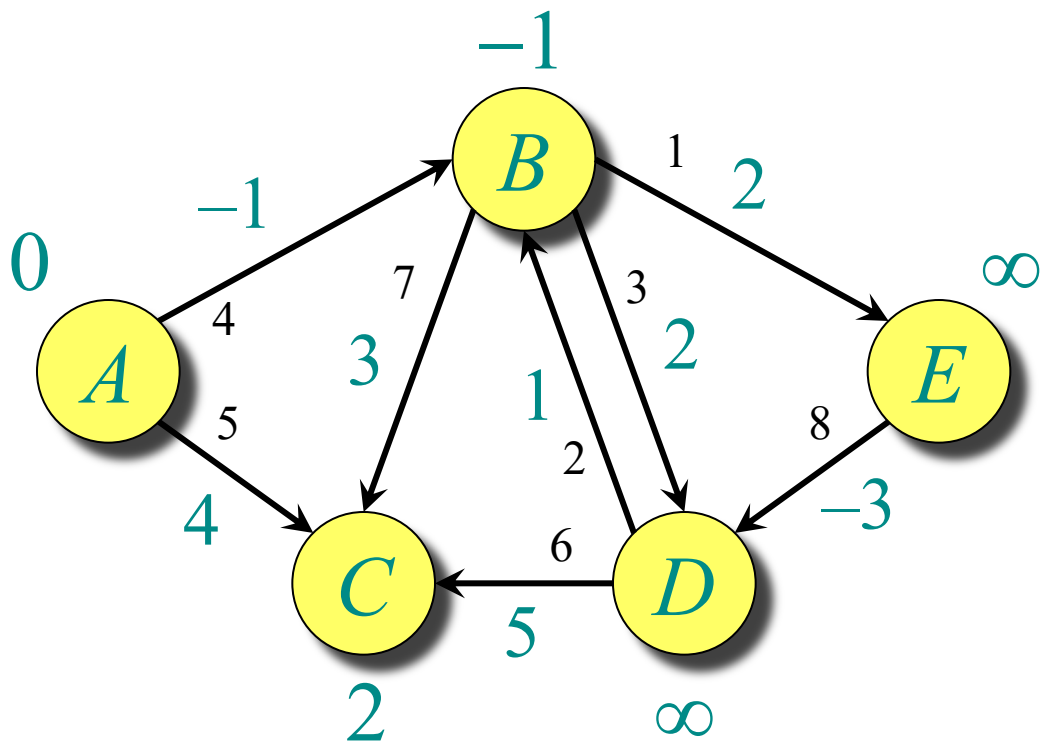
Example of Bellman-Ford



Example of Bellman-Ford

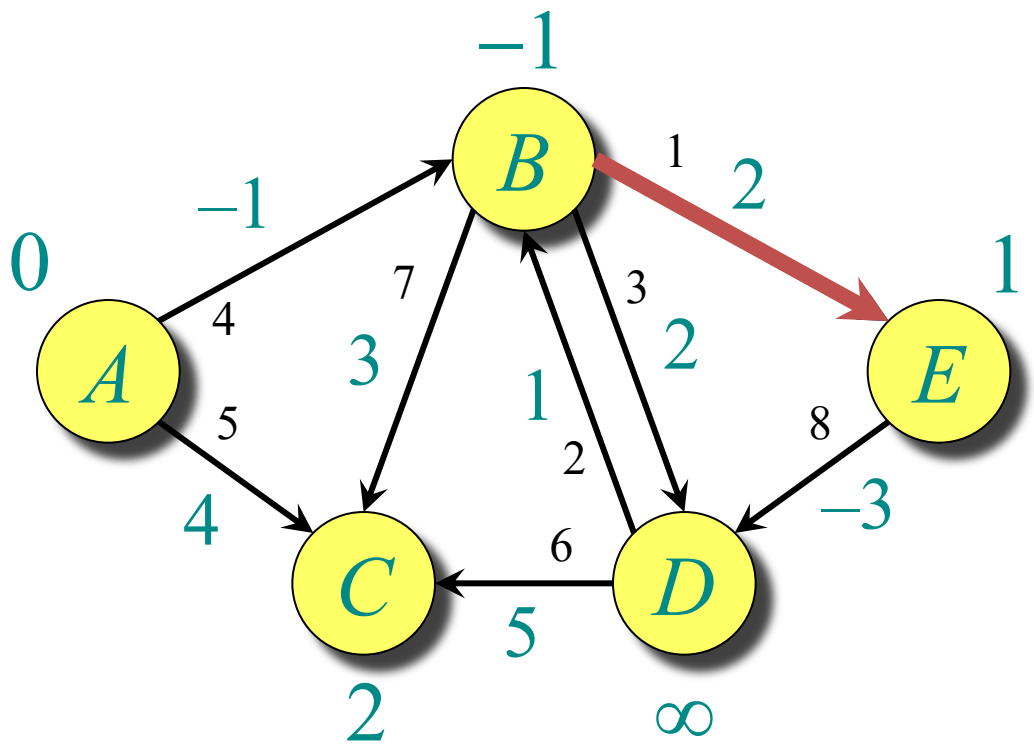


Example of Bellman-Ford

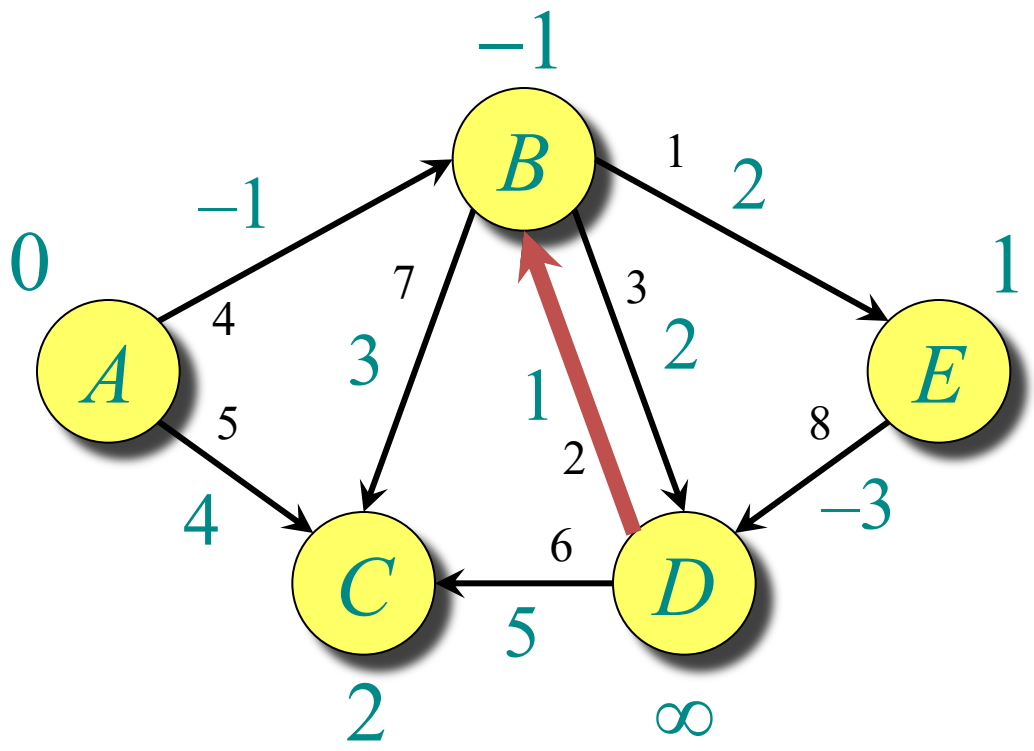


End of pass 1.

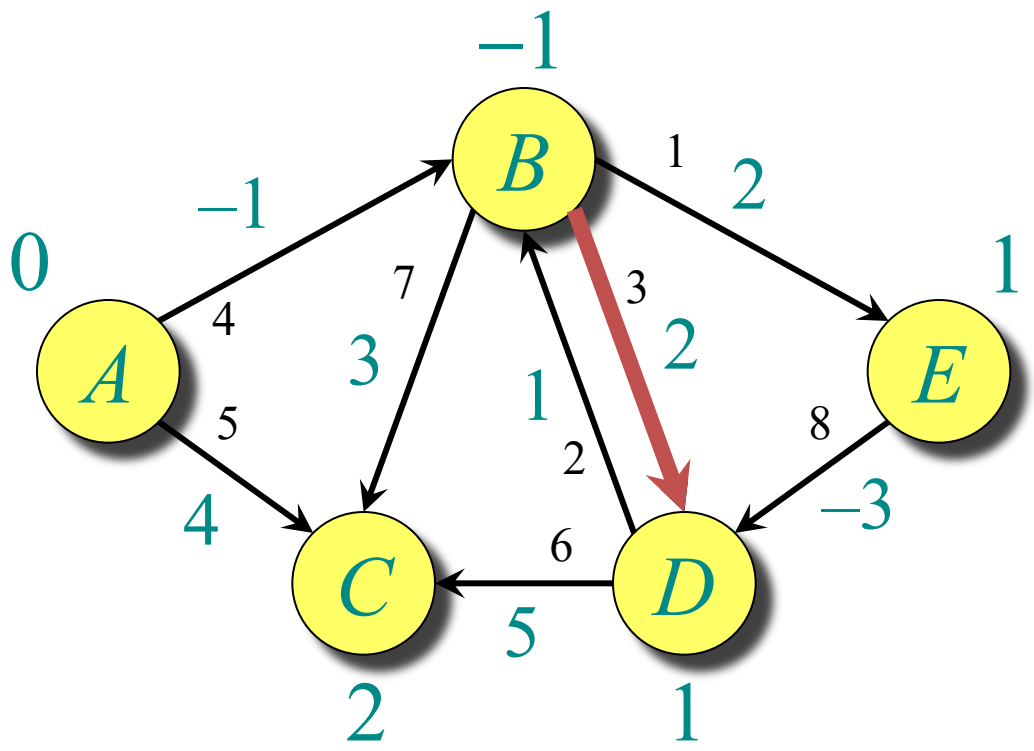
Example of Bellman-Ford



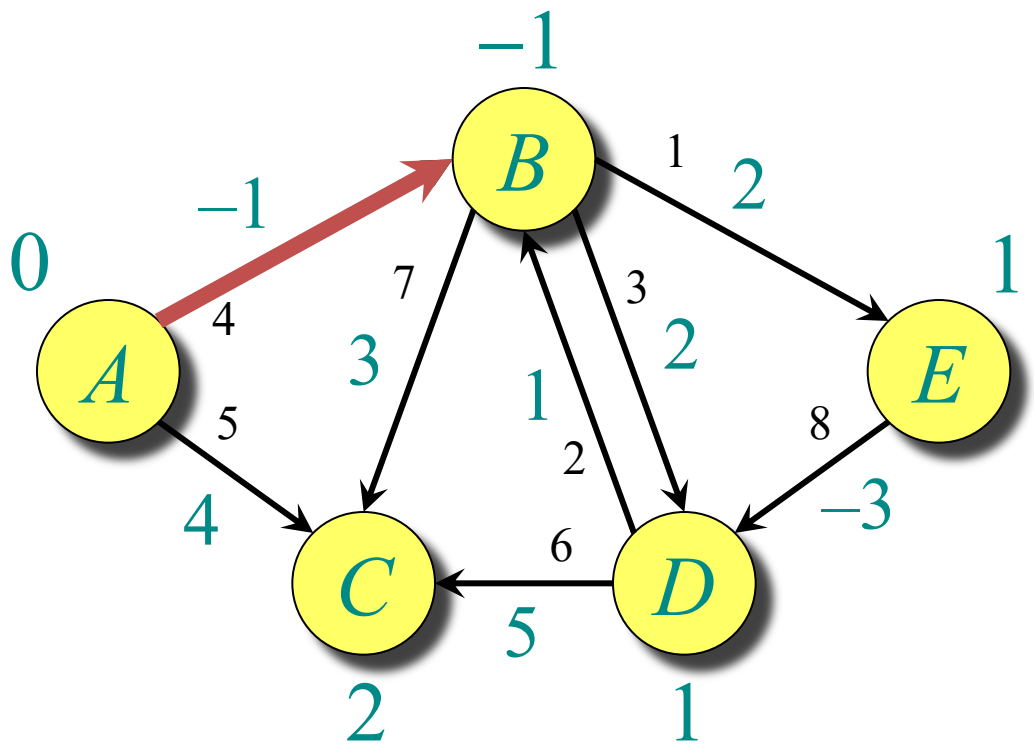
Example of Bellman-Ford



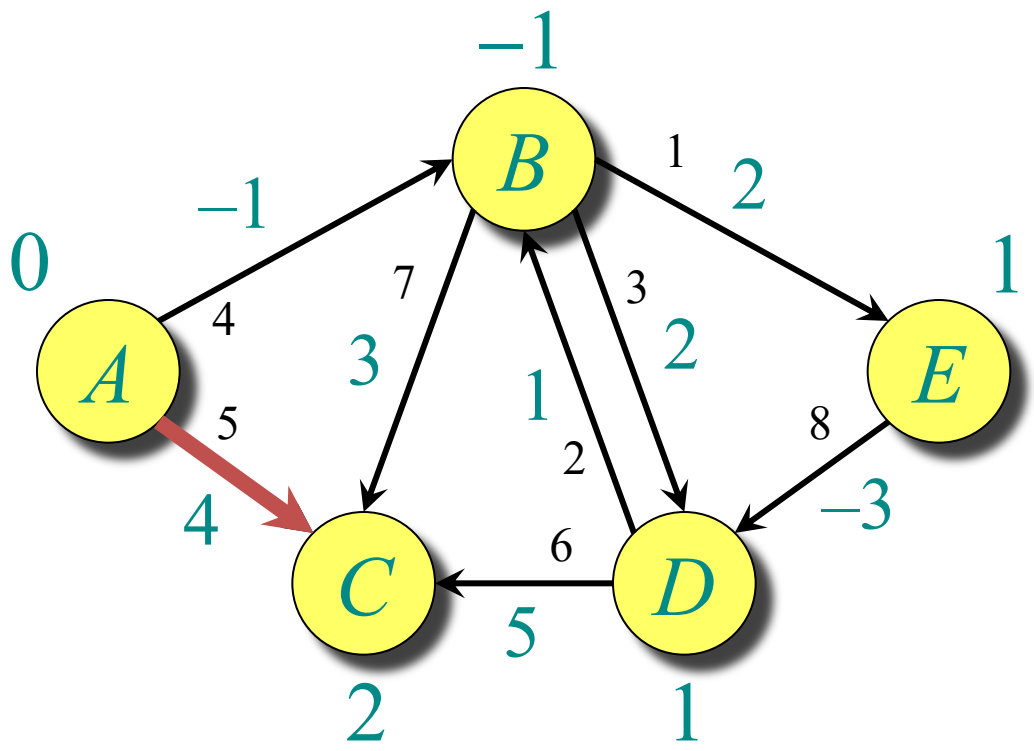
Example of Bellman-Ford



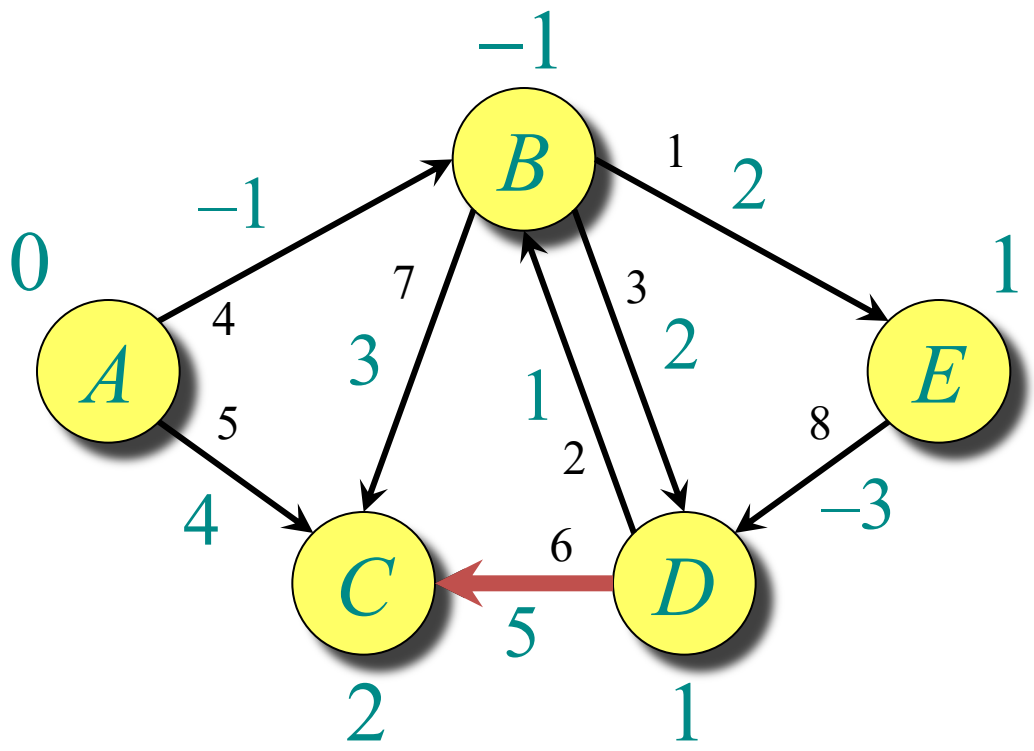
Example of Bellman-Ford



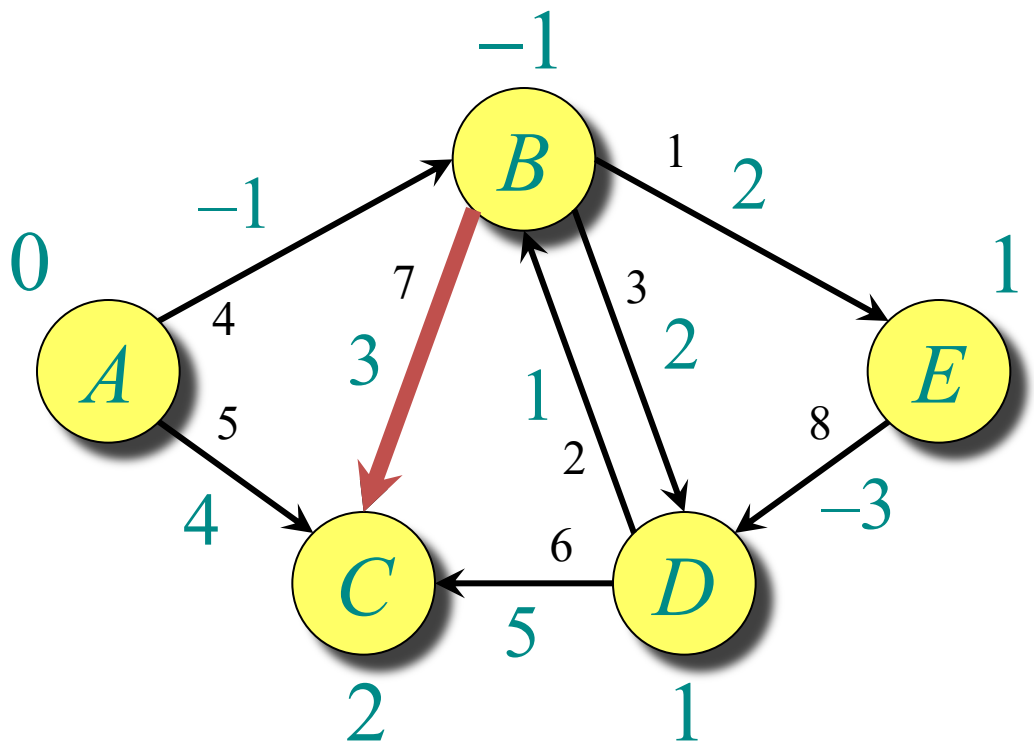
Example of Bellman-Ford



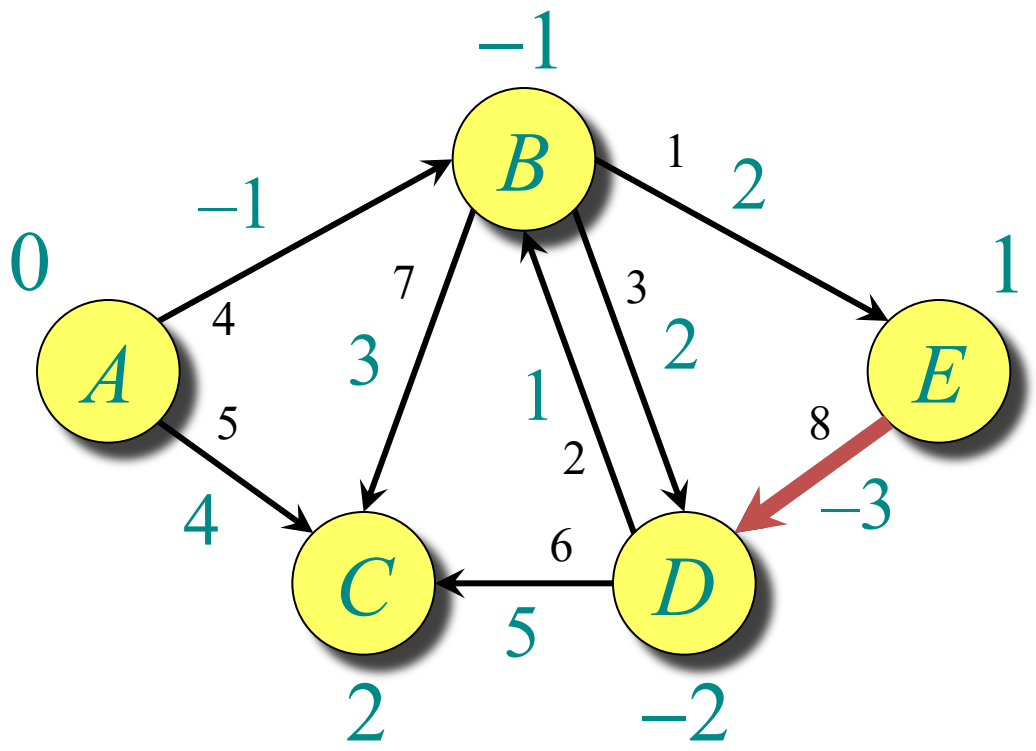
Example of Bellman-Ford



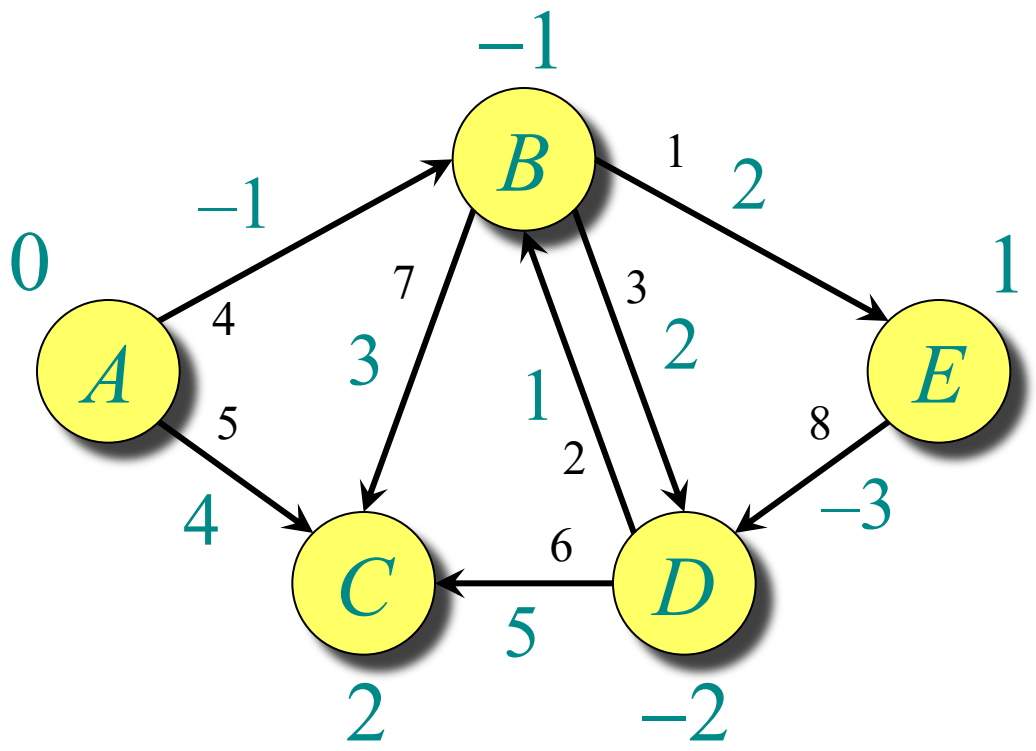
Example of Bellman-Ford



Example of Bellman-Ford



Example of Bellman-Ford



End of pass 2 (and 3 and 4).

Shortest paths

- **Single-source shortest paths**
- Nonnegative edge weights
  - Dijkstra's algorithm: $O(E + V \lg V)$
- General
  - Bellman-Ford algorithm: $O(VE)$
- DAG
  - One pass of Bellman-Ford: $O(V + E)$

Shortest paths

Single-source shortest paths

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

All-pairs shortest paths

- Nonnegative edge weights
  - Dijkstra's algorithm $|V|$ times: $O(VE + V^2 \lg V)$
- General
  - Three algorithms today.

All-pairs shortest paths

Input: Digraph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, with edge-weight function $w : E \to \mathsf{R}$.

Output: $n \times n$ matrix of shortest-path lengths $\delta(i, j)$ for all $i, j \in V$.

All-pairs shortest paths

Input: Digraph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, with edge-weight function $w : E \to \mathbb{R}$.

Output: $n \times n$ matrix of shortest-path lengths $\delta(i, j)$ for all $i, j \in V$.

IDEA:

- Run Bellman-Ford once from each vertex.
- Time $= O(V^2E)$.
- Dense graph (n^2 edges) $\Rightarrow \Theta(n^4)$ time in the worst case.

All Pairs Shortest Path: Floyd-Warshall

- A dynamic programming approach to solving
- Dist(A,B,k) = Distance from A to B going through only nodes 1..k
  - Dist(A,B,k) = 0 if A=B
  - Dist(A,B,-1) = weight(A,B) for edge from A to B (and infinity if no such edge)
  - Dist(A,B,k) = min of:
    - Dist(A,B,k-1)
    - Dist(A,k,k-1) + Dist(k,B,k-1)

- Easy to implement (4 lines of code if adjacency matrix is used)

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 $c_{ij} > c_{ik} + c_{kj}$
        then $c_{ij} \leftarrow c_{ik} + c_{kj}$ } ***relaxation***

Notes:

- Okay to omit superscripts, since extra relaxations can't hurt.
- Runs in $\Theta(n^3)$ time.
- Simple to code.
- Efficient in practice.

Recursive definition

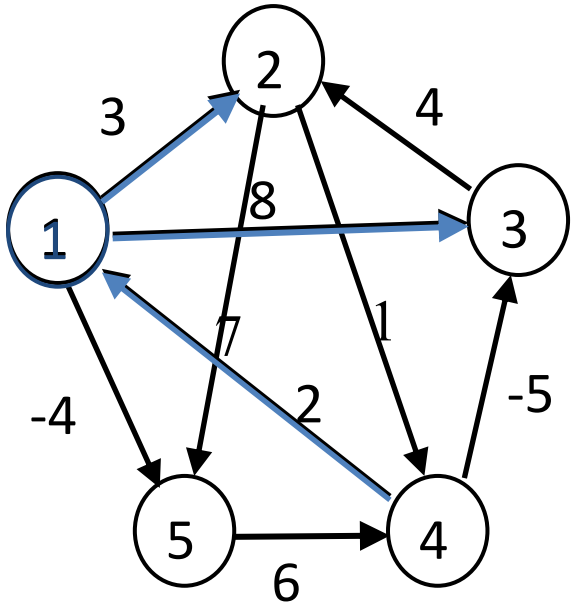
Let $d^{(k)}_{ij}$ be the weight of a shortest path from vertex *i* to vertex *j* for which all intermediate vertices are in the set {1,2,…,*k*}. Then:

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1 \end{cases}$$

The matrices $D^{(k)} = (d^{(k)}_{ij})$ will keep the intermediate solutions.

Floyd-Warshall Example

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$



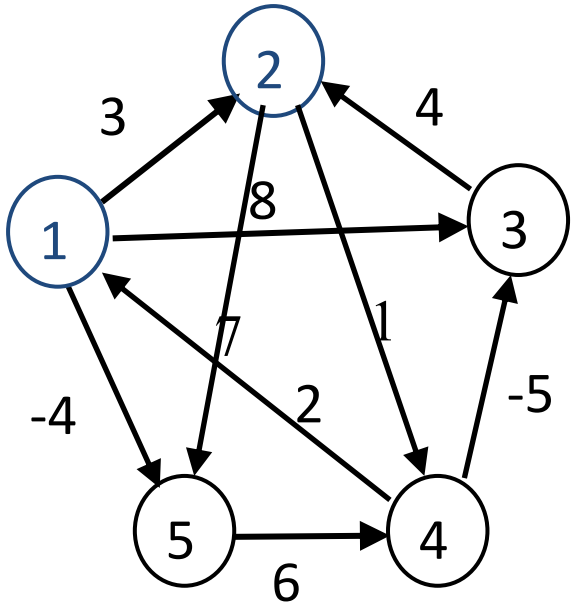
$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

K={1}

Improvements: (4,2) and (4,5)

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

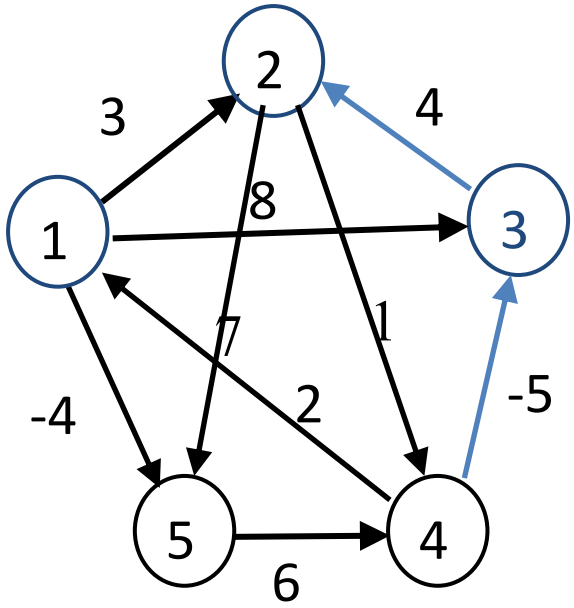
$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$



K={1,2}

Improvements: (1,4) (3,4),(3,5)

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$



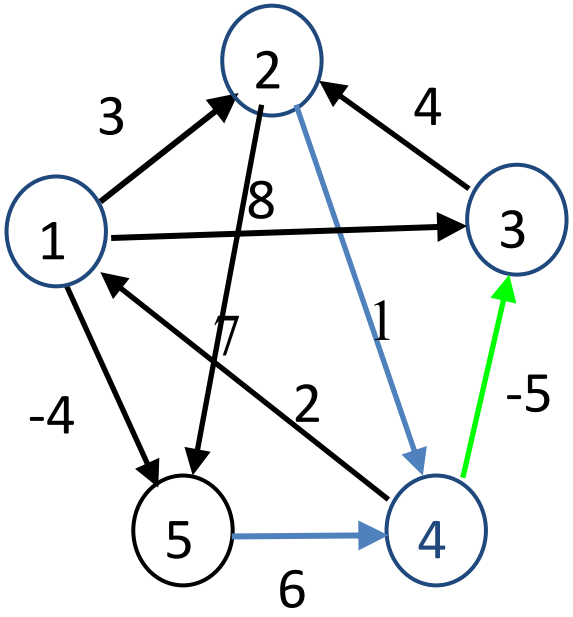
$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

K={1,2,3}

Improvement: (4,2)

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

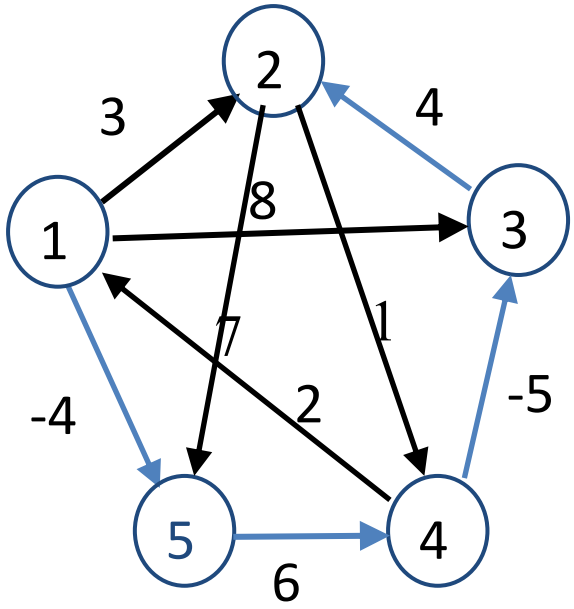


K={1,2,3,4}

Improvements: (1,3),(1,4) (2,1), (2,3), (2,5) (3,1),(3,5), (5,1),(5,2),(5,3)

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$



K={1,2,3,4,5}

Improvements:
(1,2),(1,3),(1,4)

All pairs shortest paths algorithms

- When no negative edges
  - Using Dijkstra's algorithm: $O(V^3)$
  - Using Binary heap implementation: $O(VE \lg V)$
- When no negative cycles
  - Floyd-Warshall: $O(V^3)$ time
- When negative cycles
  - Using Bellman-Ford algorithm: $O(V^2 E) = O(V^4)$
  - Johnson: $O(VE + V^2 \log V)$ time based on a clever combination of Bellman-Ford and Dijkstra