CS590/CPE590

Shortest Path Algorithms

Kazi Lutful Kabir

Spring 2023

Shortest-Path

- Given a graph (directed or undirected) $G = (V, E)$ with weight function w: $E \rightarrow \mathbf{R}$ and a vertex $s \in V$, find for all vertices $v \in V$ the minimum possible weight for path from s to v.

- The weight of path $p = v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$ is

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

- Shortest path = a path of the minimum weight

- Algorithm will compute a shortest-path tree.

Shortest-Path Problems

- Shortest-Path problems
  - **Single-Source (Single-Destination):** Find a shortest path from a given source (vertex *s*) to each of the vertices.
  - **Single-Pair:** Given two vertices, find a shortest path between them. Solution to single-source problem solves this problem efficiently, too.
  - **All-Pairs:** Find shortest-paths for every pair of vertices. Dynamic programming algorithm.

Single-Source Shortest Path

- Given a graph (directed or undirected) $G = (V, E)$ with weight function $w: E \rightarrow \mathbf{R}$ and a vertex $s \in V$, find for all vertices $v \in V$ the minimum possible weight for path from s to v.

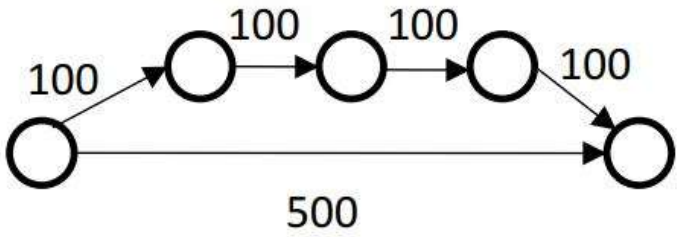
- We will discuss two general case algorithms:
  - **Dijkstra's Algorithm** (non-negative edge weights only)
  - **Bellman-Ford Algorithm** (positive and negative edge weights)

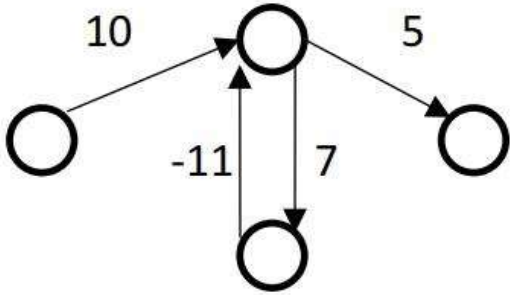
- If all edge weights are equal (let's say 1), the problem is solved by BFS in $\Theta(V + E)$ time. That's why BFS works for undirected graphs as well if the edges are of equal weight or unweighted graph.

BFS no longer Works!

Why BFS won't work: Shortest path may not have the fewest edges

- Annoying when this happens with costs of flights



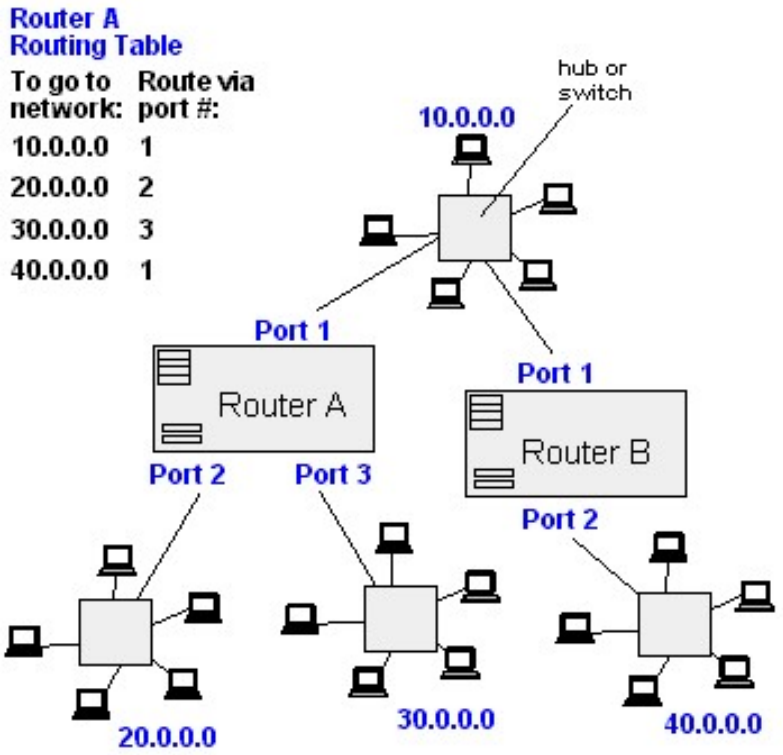


We will assume there are no negative weights

- *Problem* is *ill-defined* if there are negative-cost *cycles*

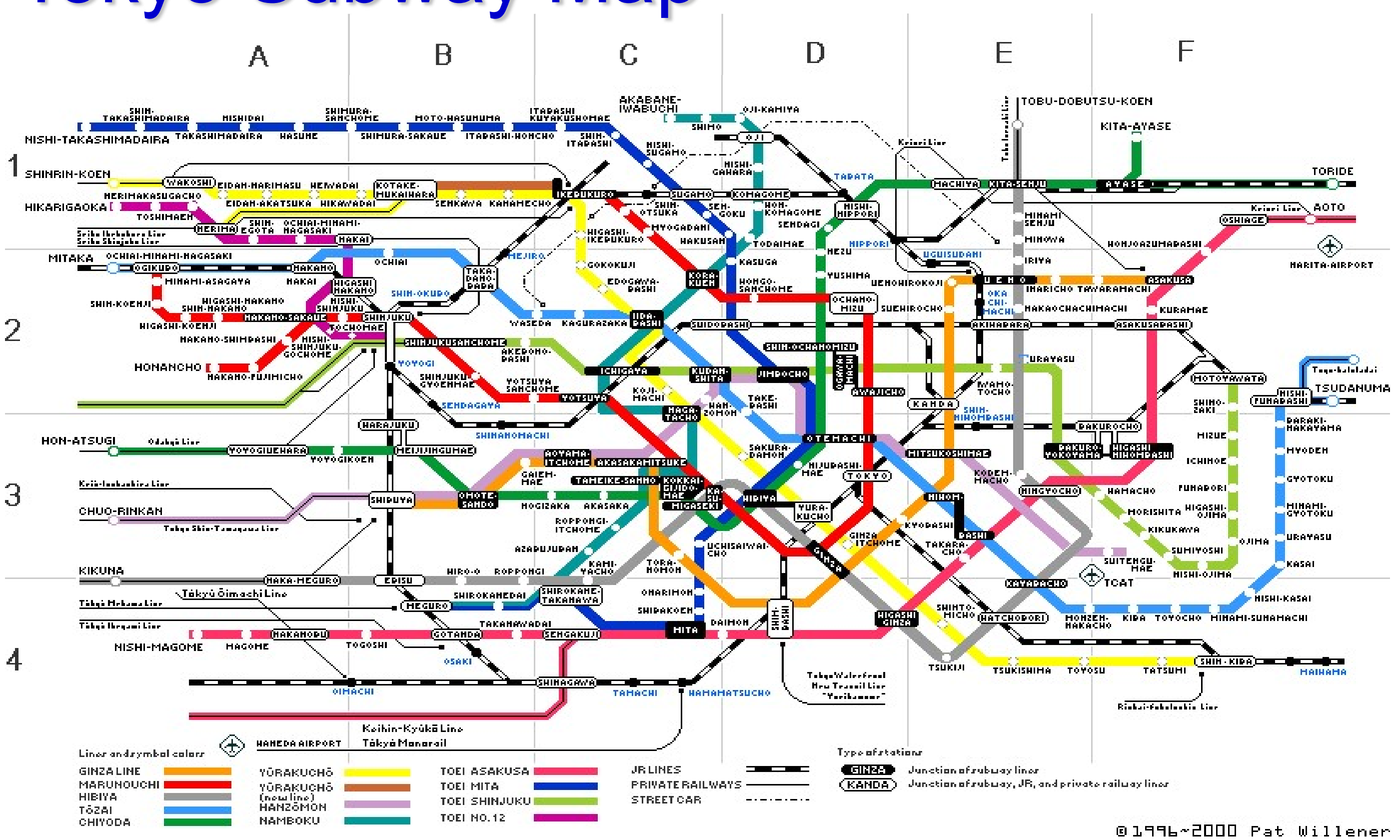
Single-Source Shortest Path

- Traffic Information Systems are most prominent use
- Mapping (Map Quest, Google Maps)
- Routing Systems
- Flight Reservations



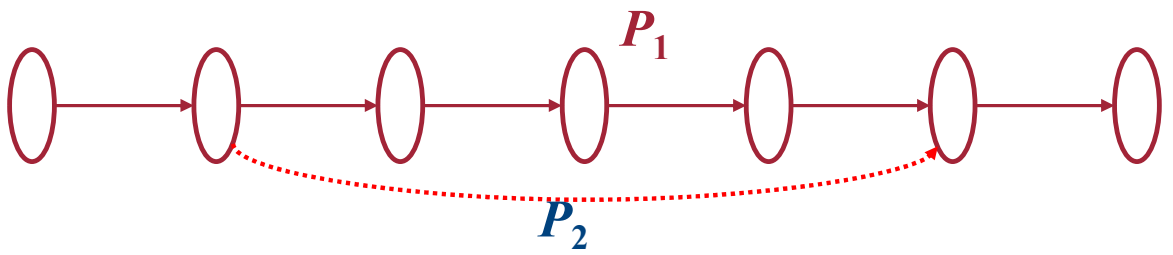
Single-Source Shortest Path

Tokyo Subway Map



Shortest Path Properties

- The shortest path problem satisfies the ***optimal substructure*** *property*:
  - Sub-paths of shortest paths are shortest paths.



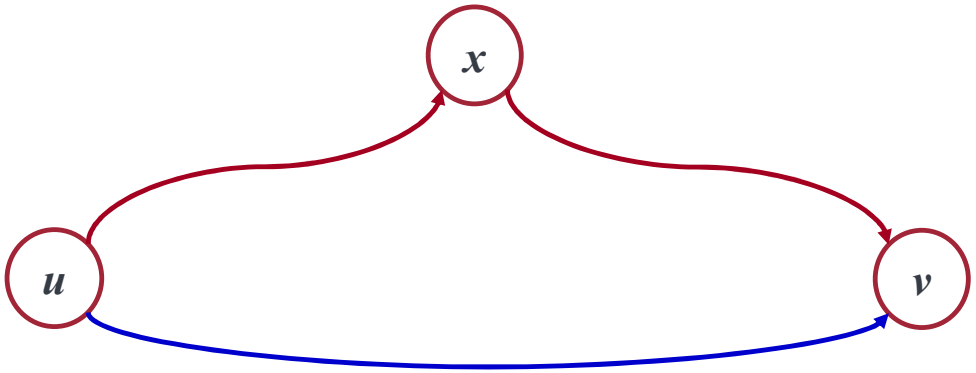
  - **Proof**: suppose some sub-path P_1 is not a shortest path
    - There must then exist a shorter sub-path P_2
    - Could substitute the sub-path P_1 by the shorter path P_2
    - But then overall path is not the shortest path. Contradiction

Shortest Path Properties

- Define $\delta(u, v)$ to be the weight of the shortest path from u to v
- Shortest paths satisfy the *triangle inequality*:

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

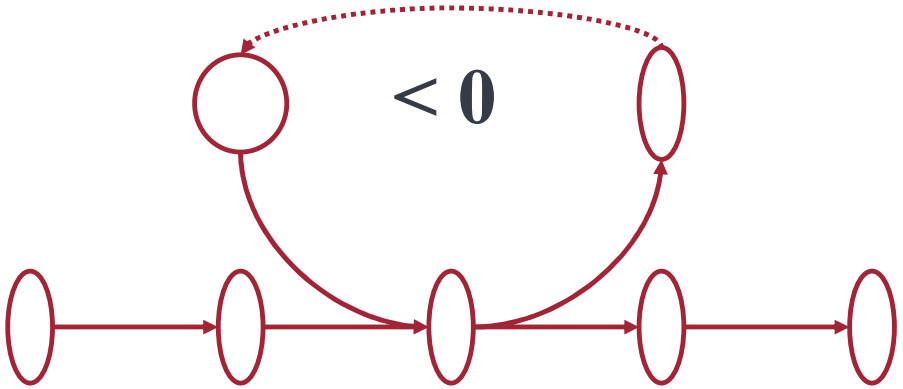
- **Proof** :



This path is no longer than any other path

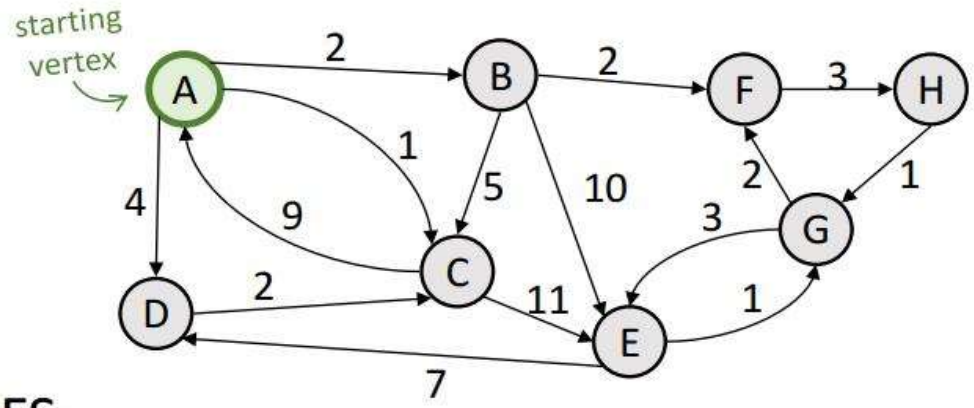
Shortest Path Properties

- In graphs with negative weight cycles, some shortest paths will not exist (*Why* ?):



Algorithm: General Idea

Goal: From one starting vertex, what are the shortest paths to each of the other vertices (for a weighted graph)?

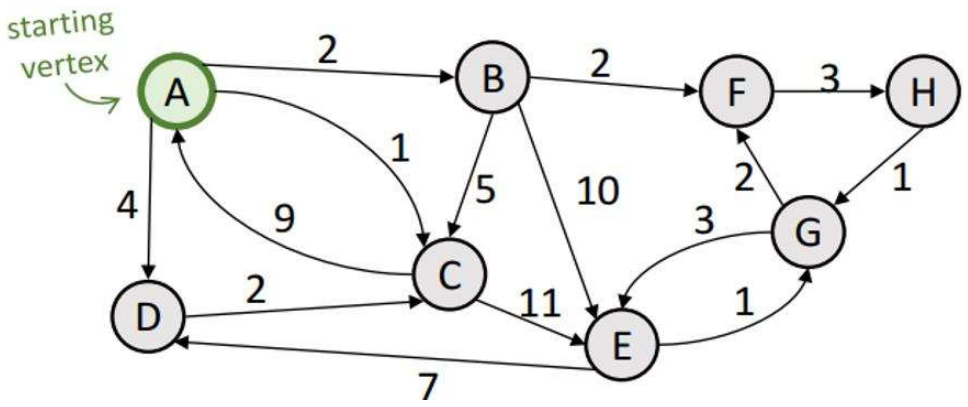


Idea: Similar to BFS

- Repeatedly increase a "set of vertices with known shortest distances"
- Any vertex not in this set will have a "best distance so far"
- Each vertex has a "cost" to represent these shortest/best distances
- Update costs (i.e. "best distances so far") as we add vertices to set

Single-Source Shortest Path Problem

Single-Source Shortest Path Problem - The problem of finding shortest paths from a source vertex *v* to all other vertices in the graph.



Dijkstra's algorithm

Dijkstra's algorithm - is a solution to the single-source shortest path problem in graph theory.
-- Named after its inventor Edsger Dijkstra (1930-2002)

Works on both directed and undirected graphs. However, all edges must have **nonnegative weights**.

Approach: Greedy

Input: Weighted graph G={E,V} and source vertex $v \in V$, such that all edge weights are **nonnegative**

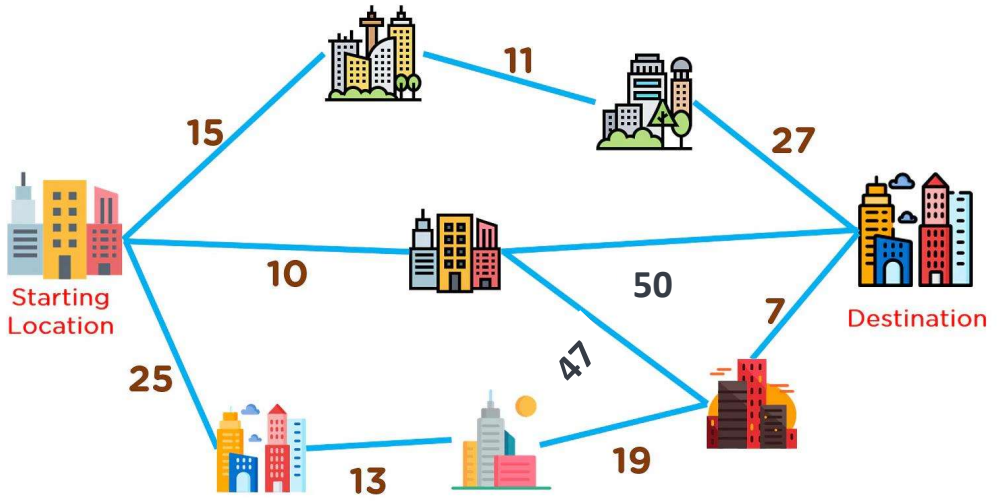
Output: Lengths of shortest paths (or the shortest paths themselves) from a given source vertex $v \in V$ to all other vertices

Greedy Algorithm

- Dijkstra's Algorithm is one example of ...
  - A greedy algorithm:
    - -- Make a locally optimal choice at each stage to (hopefully) find a global optimum
    - -- Settle on the best-looking option at each repeated step
    - -- For some problems, greedy algorithms cannot find best answer!

Greedy Algorithms: Principles

- A greedy algorithm works in phases
- At each phase:
  - You take the best you can get right now, without regard for future consequences.
  - You hope that by choosing a local optimum at each step, you will end up at a global optimum.



Dijkstra's algorithm - Pseudocode

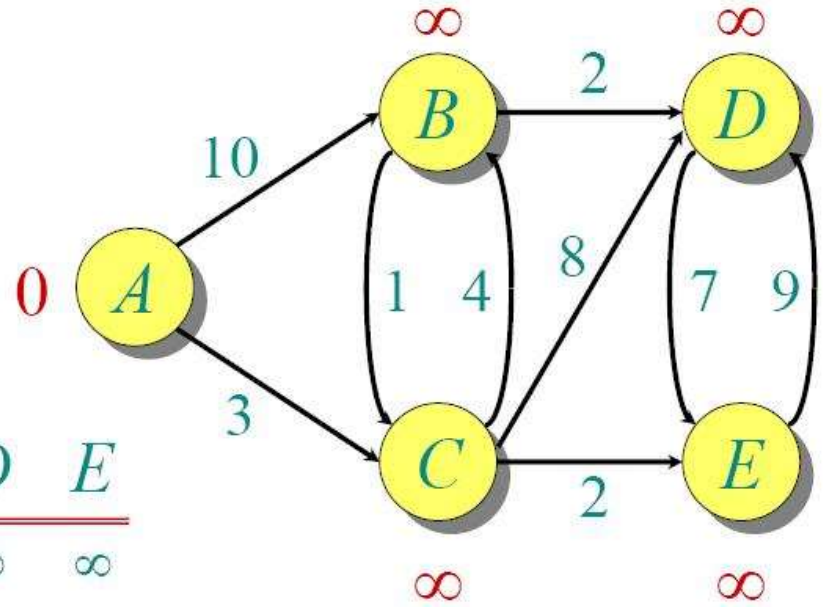
```
dist[s] ← 0                                  (distance to source vertex is zero)
for  all v ∈ V–{s}
       do  dist[v] ←∞                        (set all other distances to infinity)
S←∅                                          (S, the set of visited vertices is initially empty)
Q←V                                          (Q, the queue initially contains all vertices)
while Q ≠∅                                   (while the queue is not empty)
 do  u ← min.distance(Q,dist)                (select the element of Q with the min. distance)
     S ←S ∪ {u}                                    (add u to list of visited vertices)
      for all v ∈ Adj.[u]
            do  if   dist[v] > dist[u] + w(u, v)          (if new shortest path found)
                     then      d[v] ←d[u] + w(u, v)       (set new value of shortest path)

return dist
```

Dijkstra Example

Initialize:

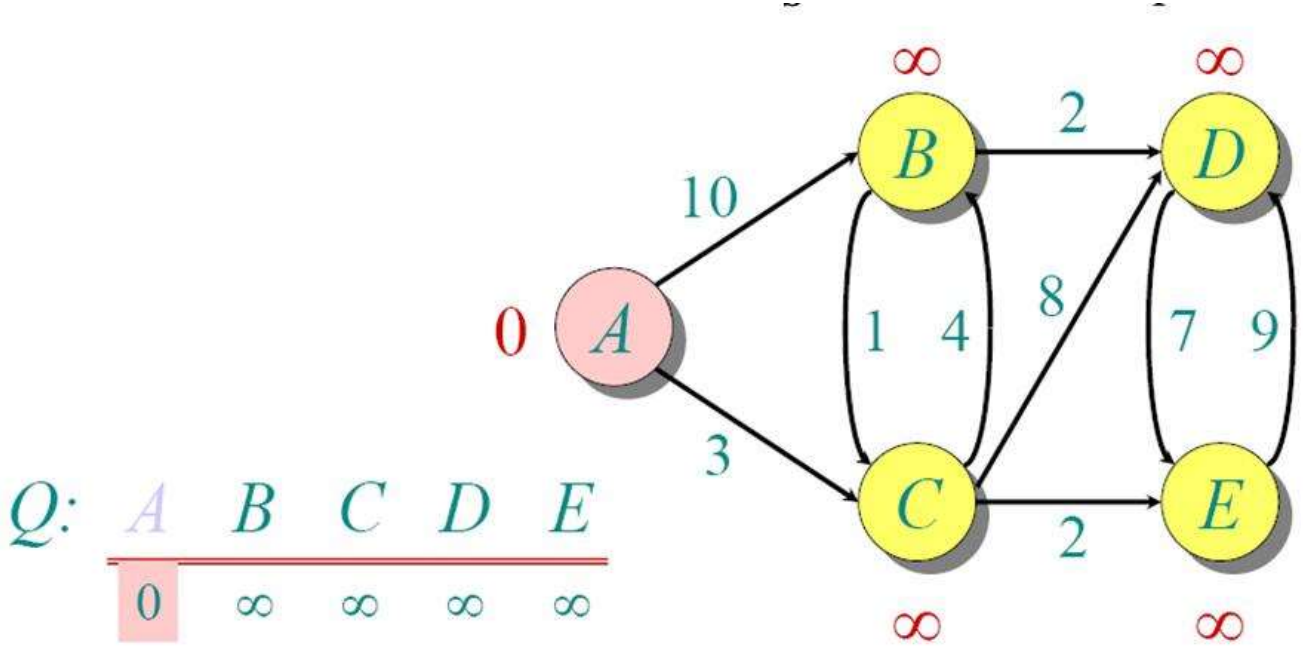
Source: **A**



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

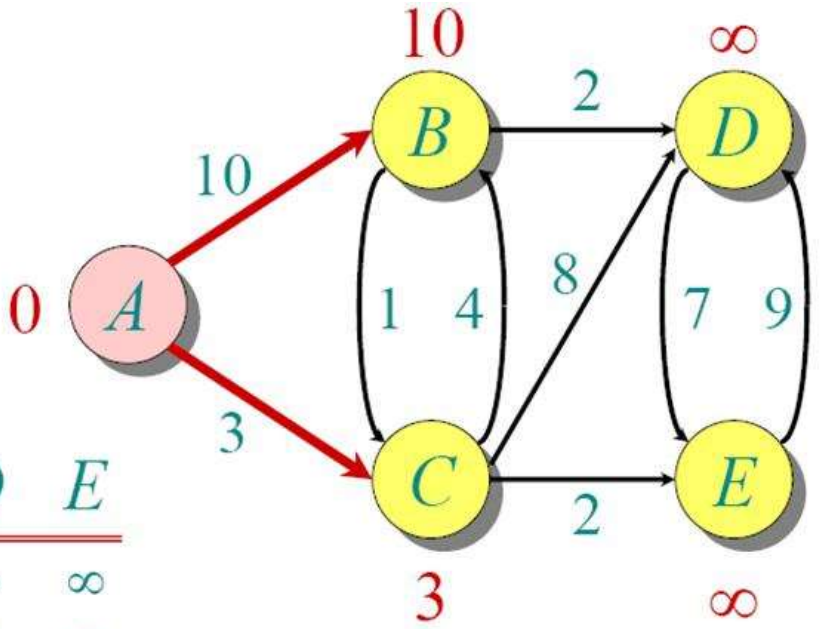
S: {}

Dijkstra Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

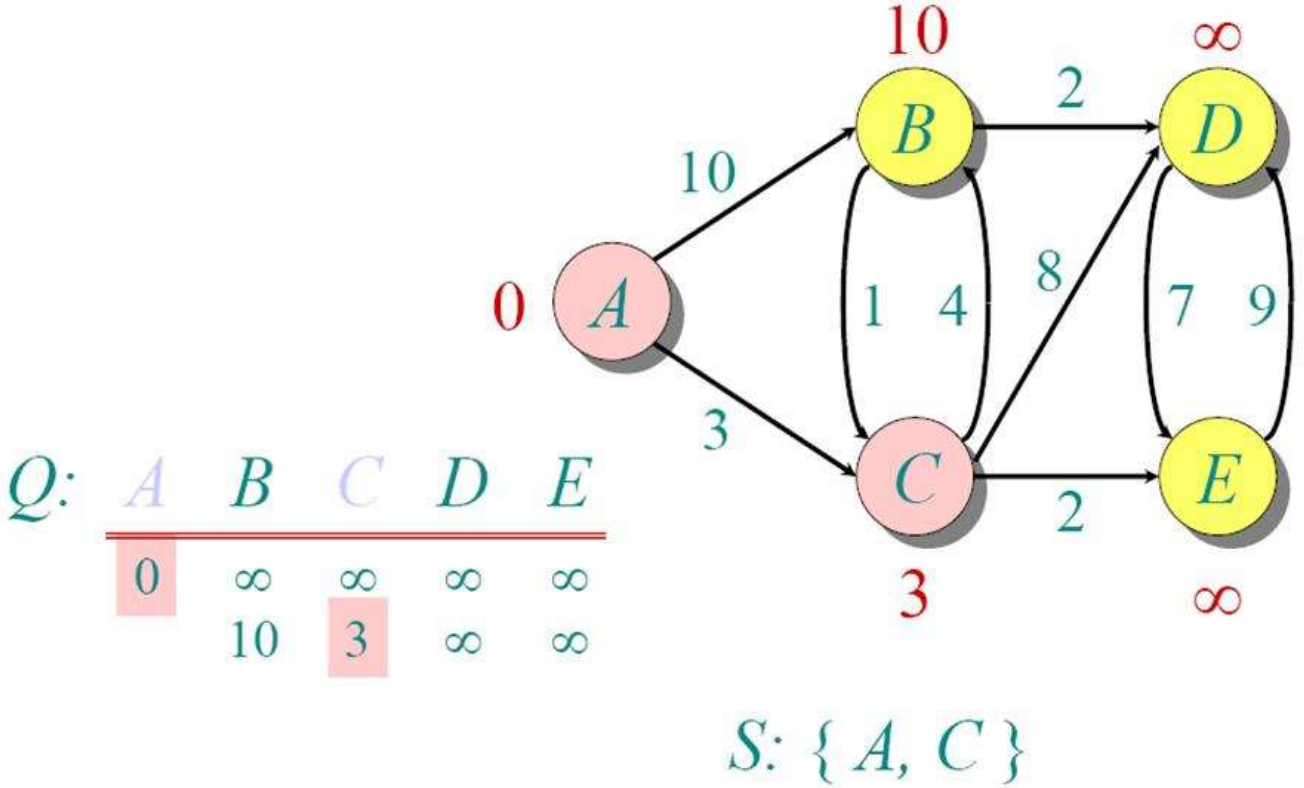
Dijkstra Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

$S: \{ A \}$

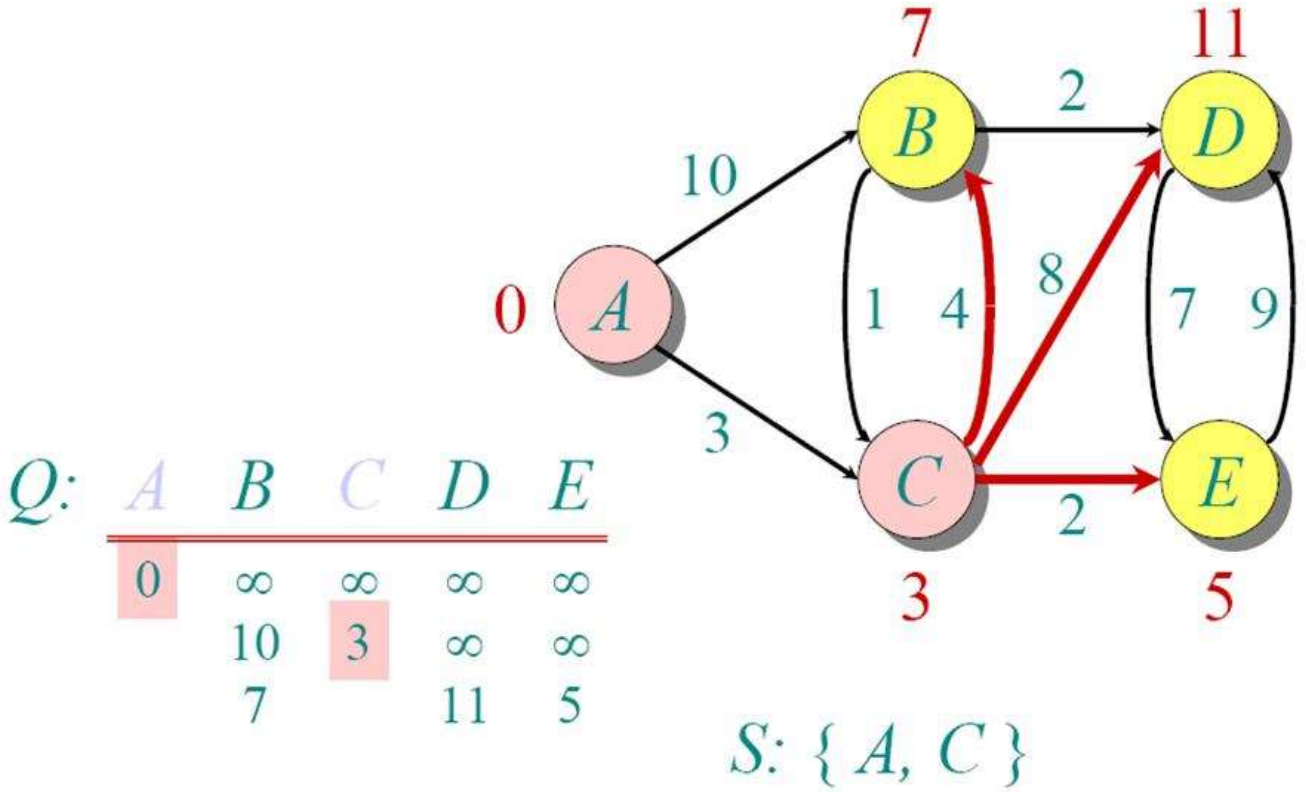
Dijkstra Example



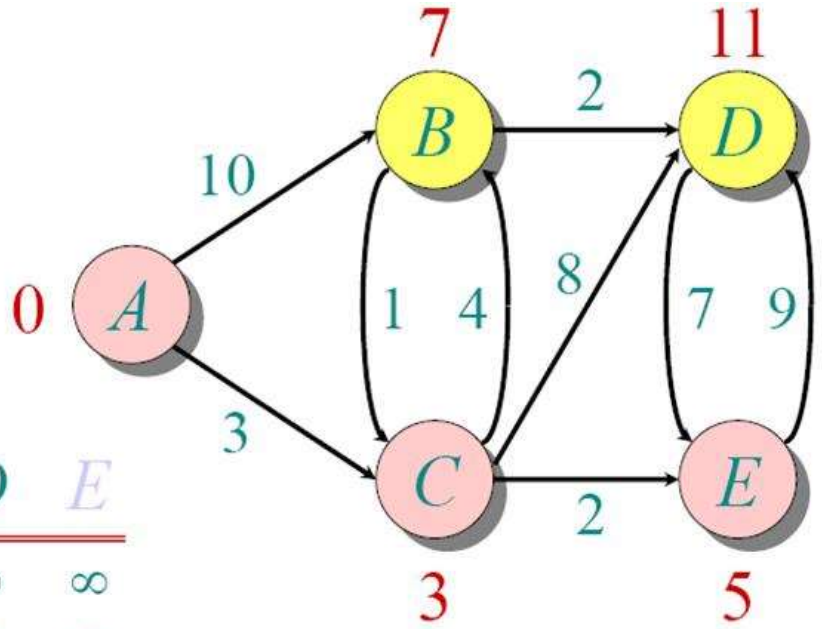
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

$S: \{ A, C \}$

Dijkstra Example



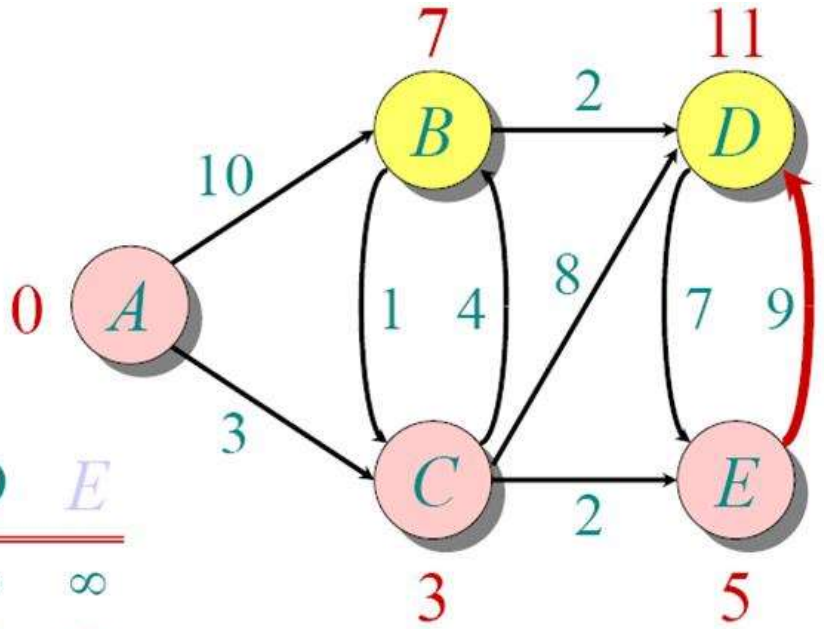
Dijkstra Example



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

$S: \{ A, C, E \}$

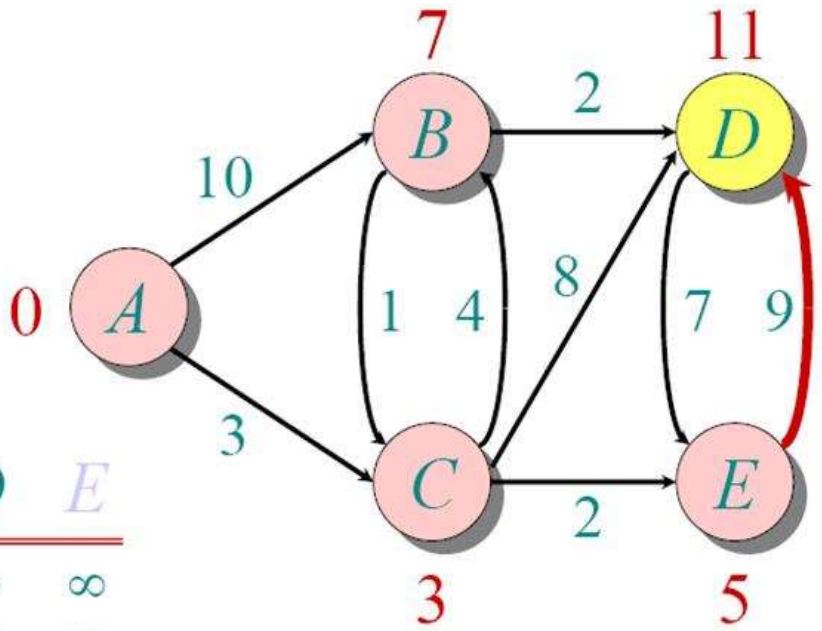
Dijkstra Example



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

$S: \{ A, C, E \}$

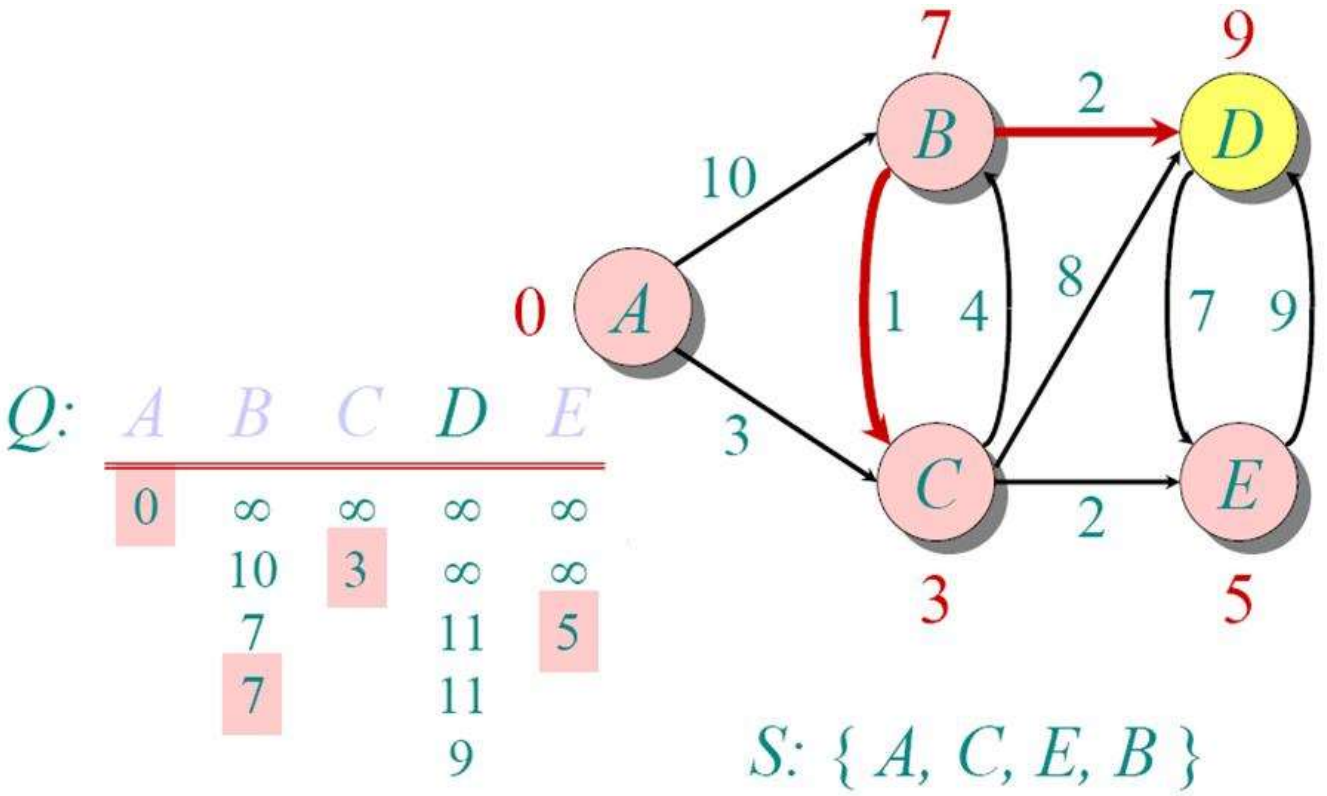
Dijkstra Example



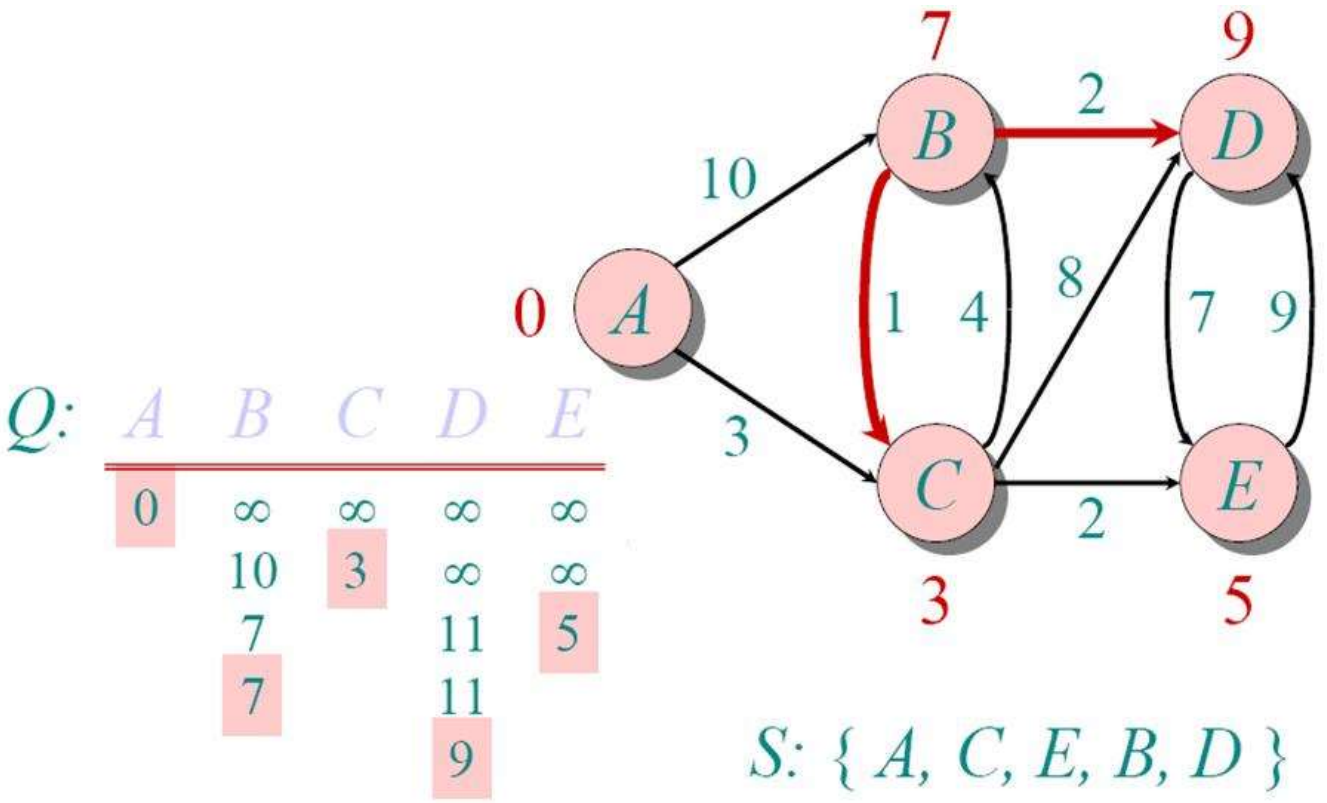
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

$S: \{ A, C, E, B \}$

Dijkstra Example



Dijkstra Example



Running Time: Using List/Array

The simplest implementation of the Dijkstra's algorithm stores vertices in an ordinary linked list or array

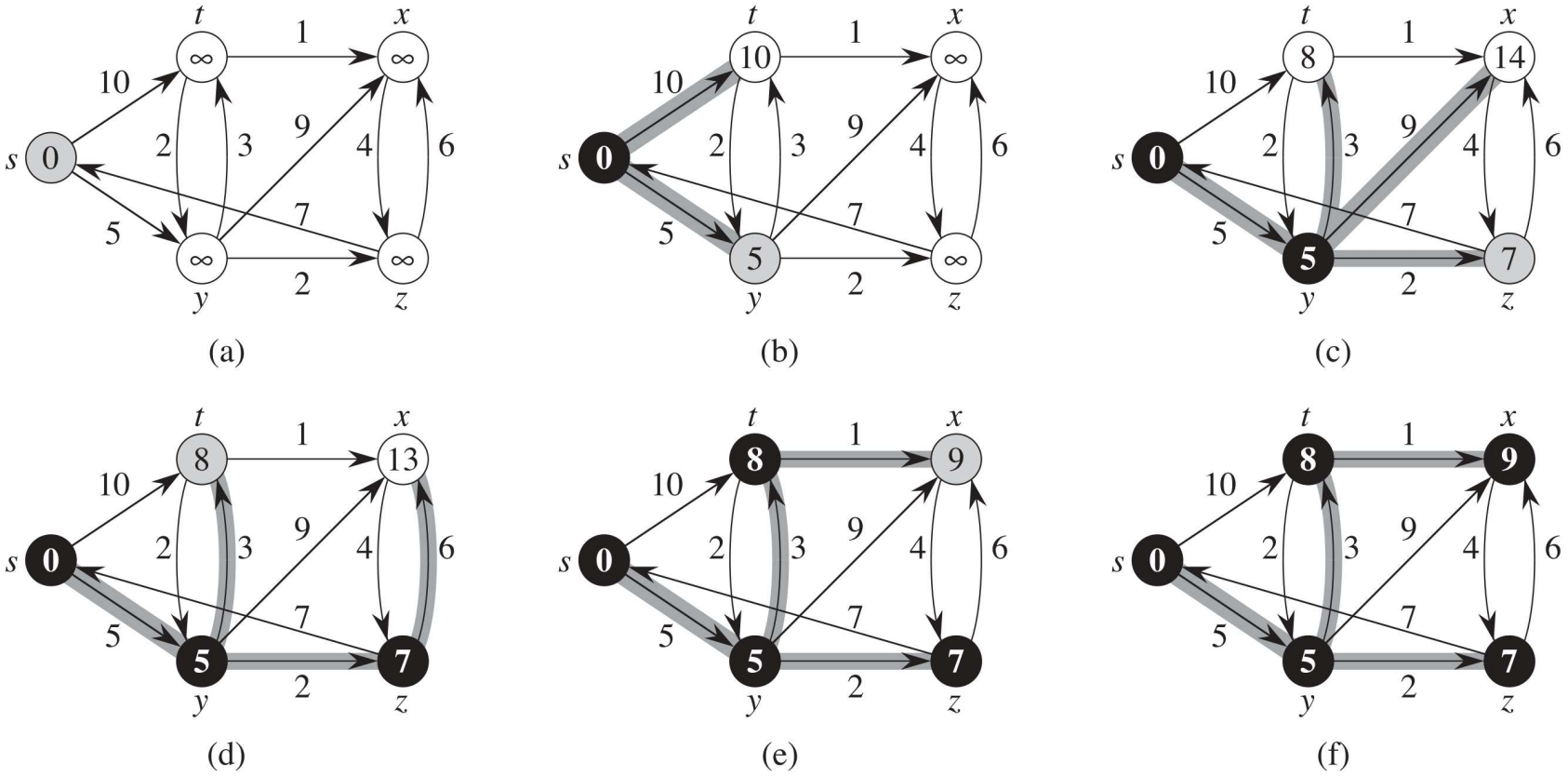
- Good for dense graphs (many edges)

- |V| vertices and |E| edges
- Initialization O(|V|)
- While loop O(|V|)
  - Find and remove min distance vertices O(|V|)
  - Potentially |E| updates
    - Update costs O(1)
- Reconstruct path O(|E|)

Total time **$O(|V^2| + |E|) = O(|V^2|)$**

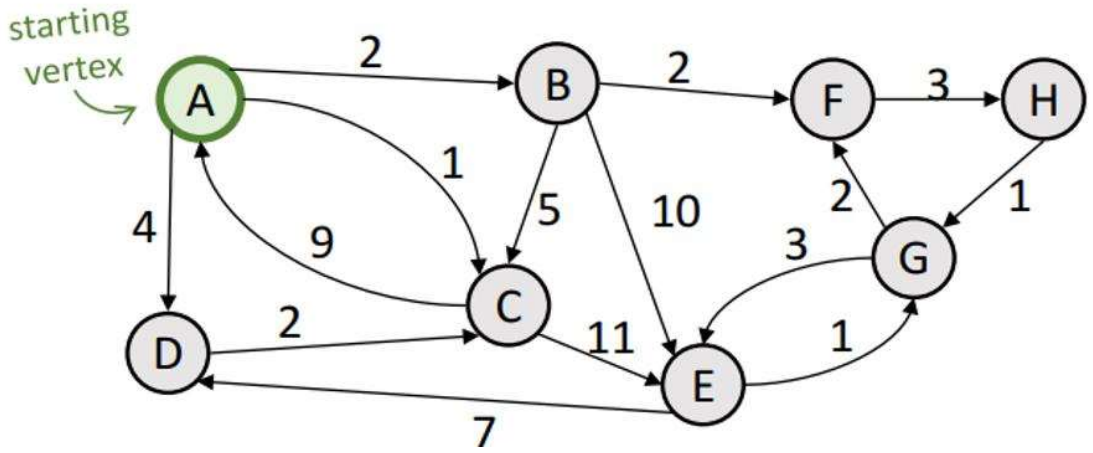
- Can be improved to **O(|V| log |V| + |E| log |V|)** using priority queue & heap data structure

Dijkstra's Algorithm: Another Example

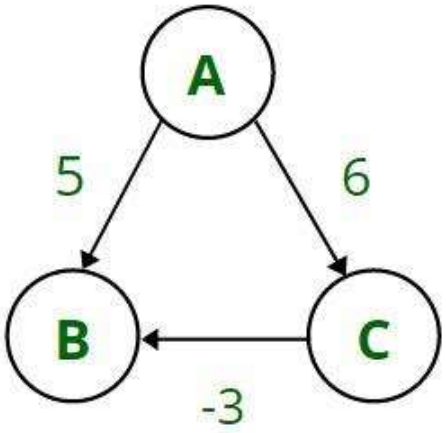


Exercise for Practice

- Try Dijkstra's Algo on the following graph:



When Dijkstra's Algo Fails?



Bellman-Ford Algorithm

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

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2       v.d = ∞
3       v.π = NIL
4  s.d = 0
```

Initialize d[], which will converge to shortest-path value δ

Relaxation: Make |V|-1 passes, relaxing each edge

Test for solution
Under what condition do we get a solution?

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2       v.d = u.d + w(u, v)
3       v.π = u
```

Bellman-Ford Algorithm

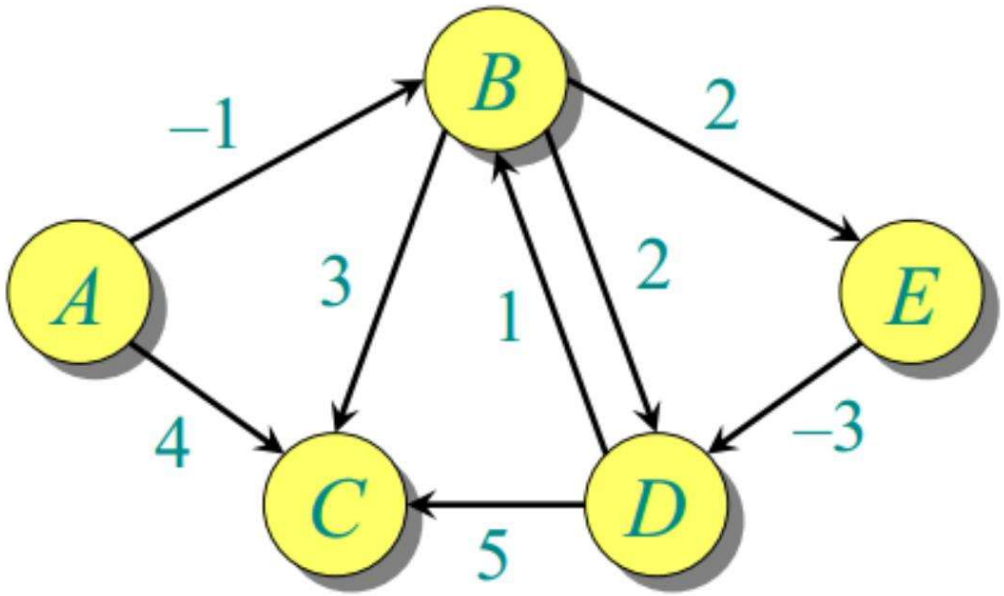
```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3       for each edge (u, v) ∈ G.E
4            RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6       if v.d > u.d + w(u, v)
7            return FALSE
8  return TRUE
```

Q: What will be the running time?

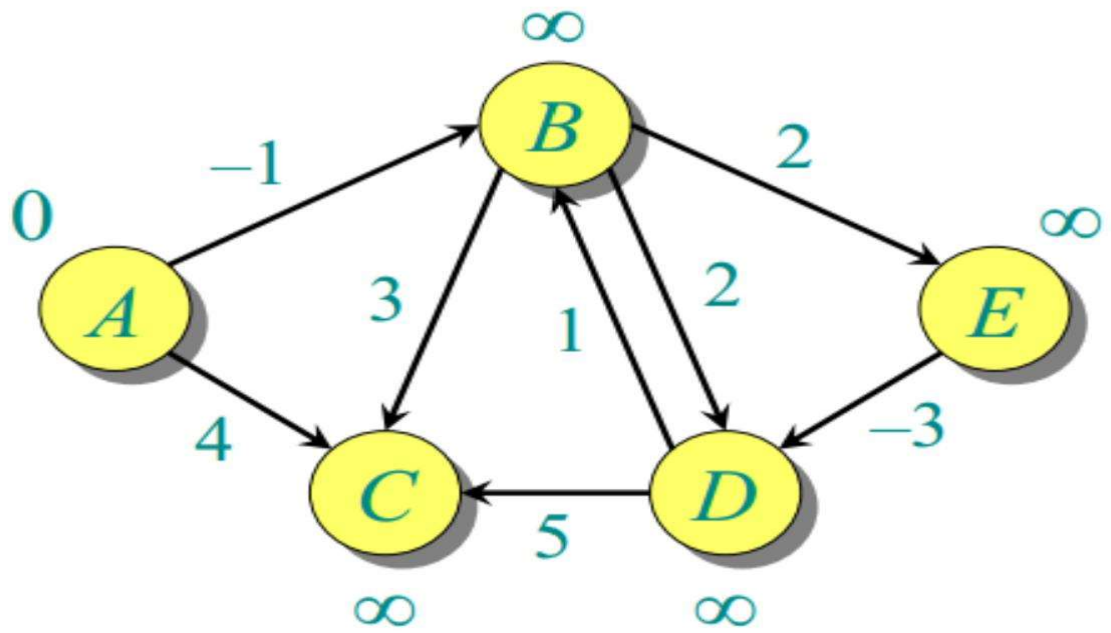
A: O(V E)

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2       v.d = u.d + w(u, v)
3       v.π = u
```

Bellman-Ford Algorithm: Example



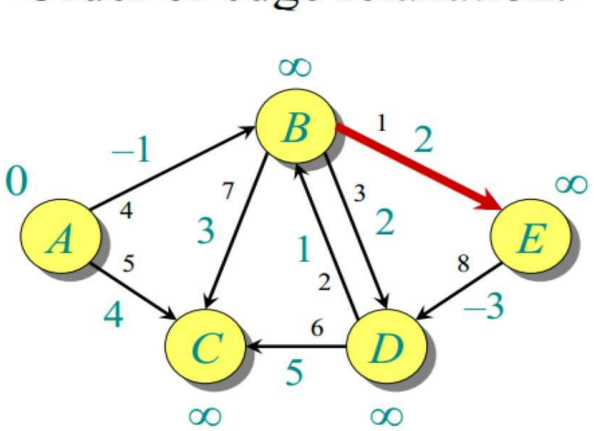
Bellman-Ford Algorithm: Example

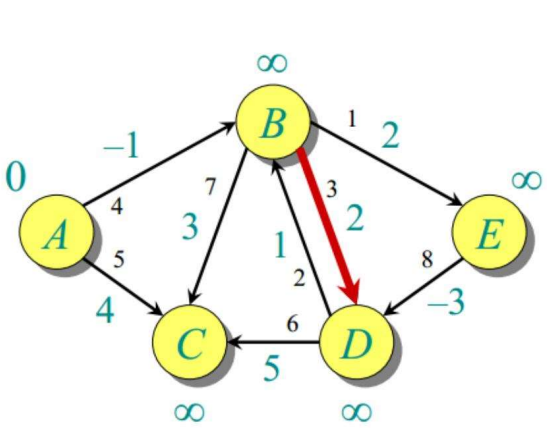


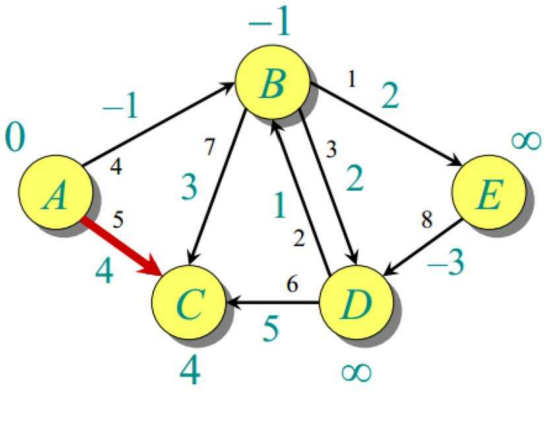
Initialization.

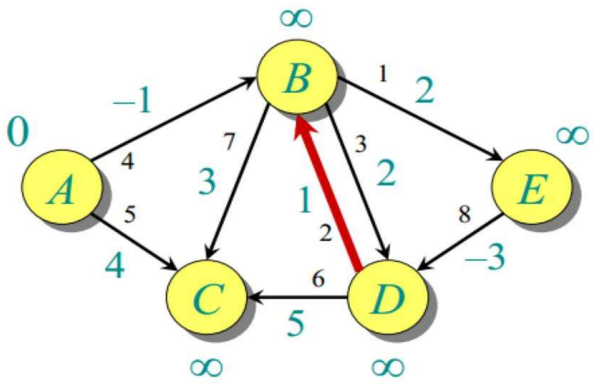
Bellman-Ford Algorithm: Example

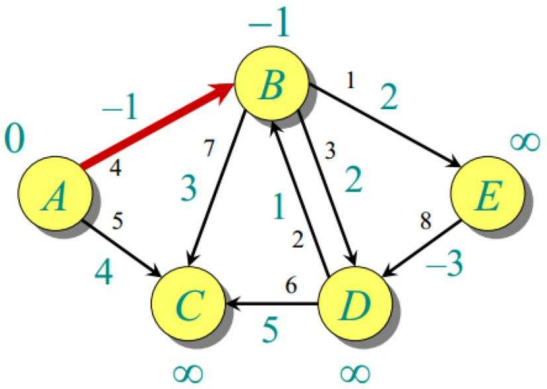
Order of edge relaxation.

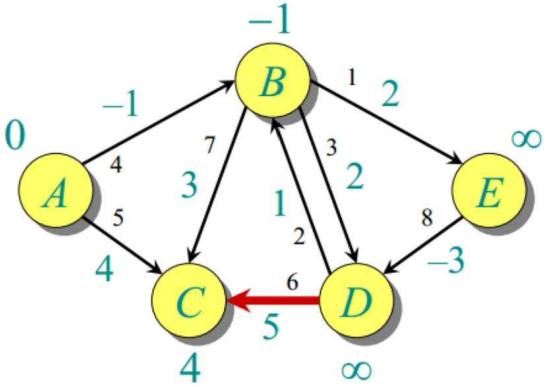




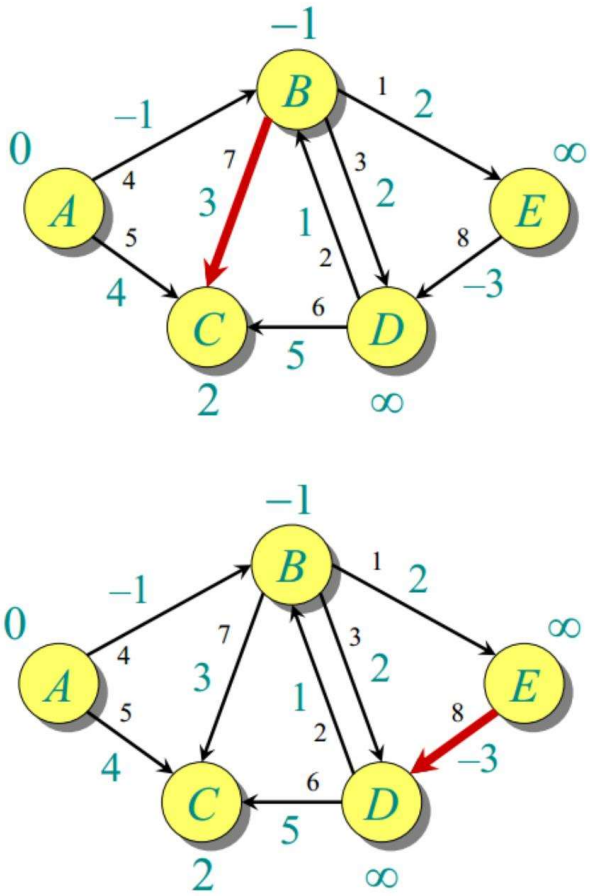


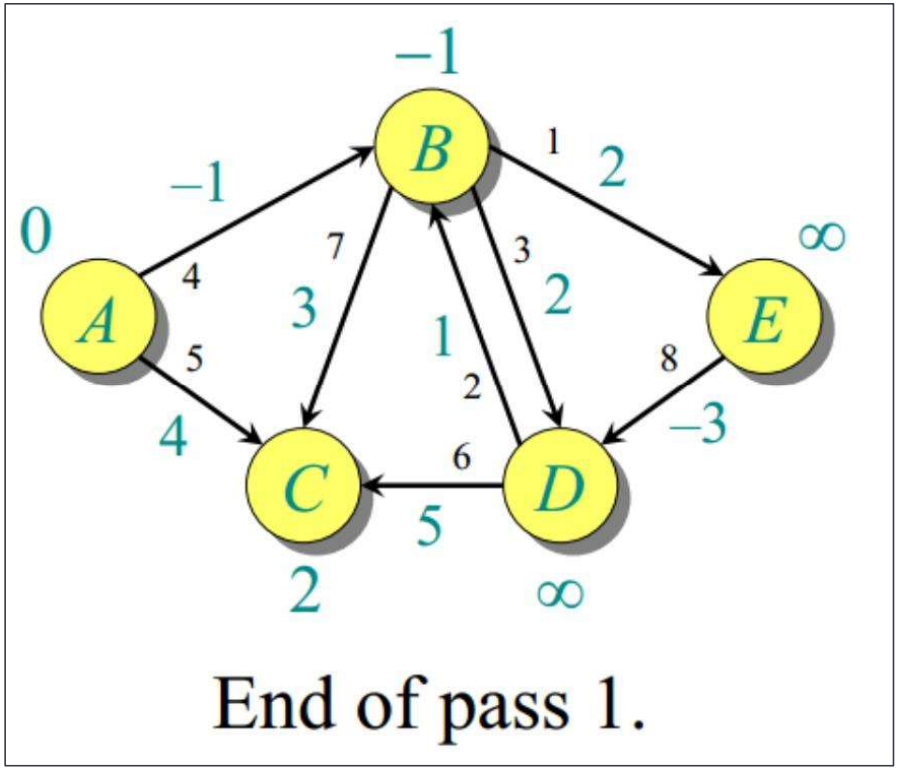




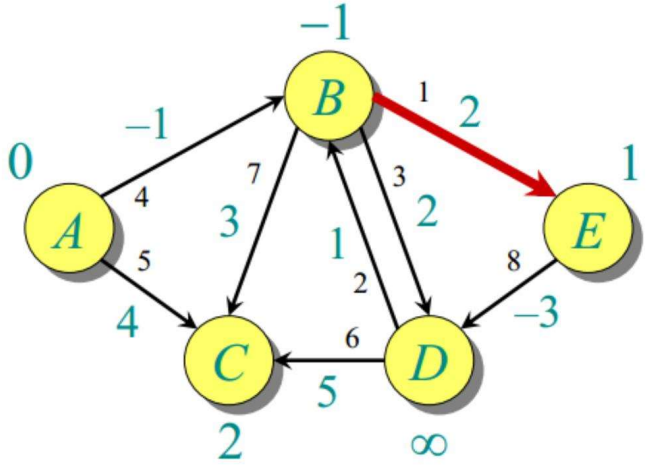


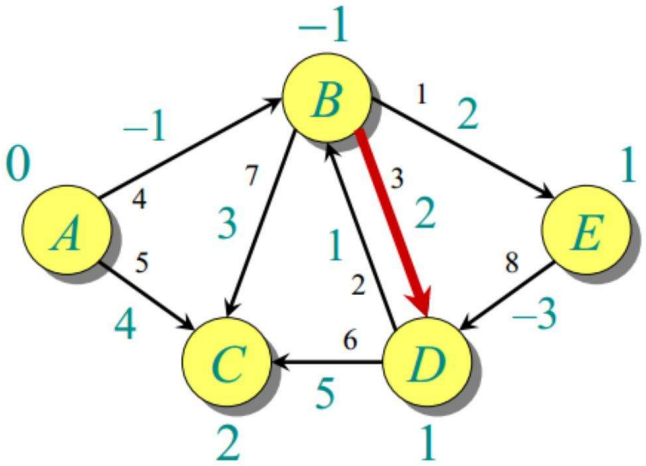
Bellman-Ford Algorithm: Example

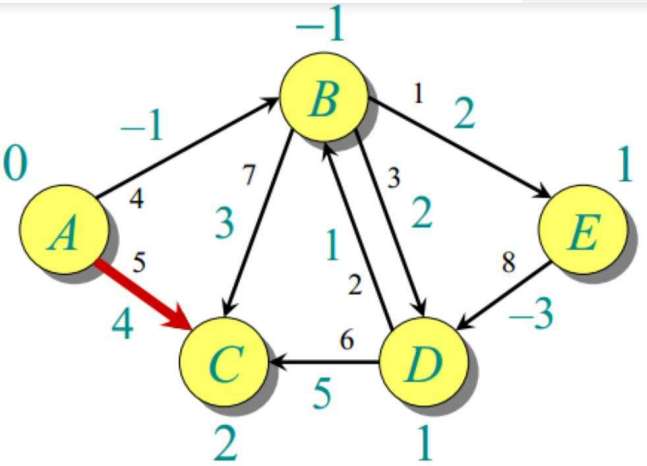


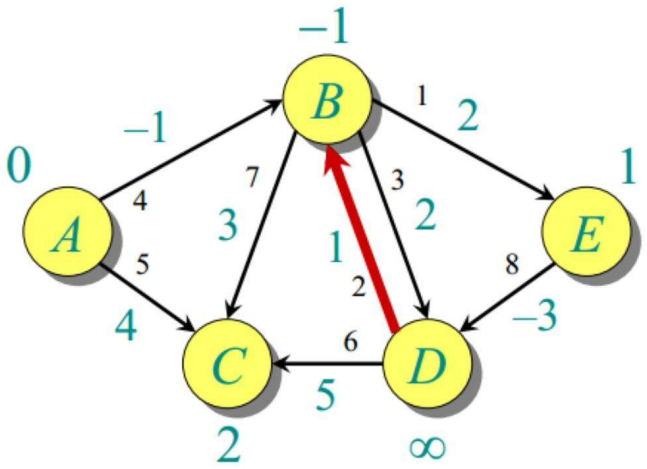


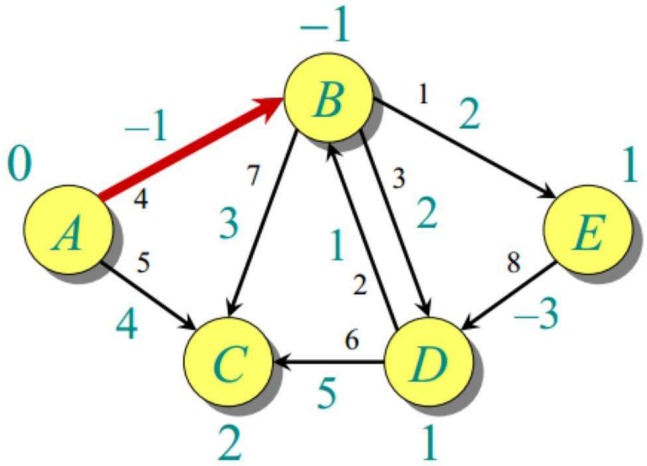
Bellman-Ford Algorithm: Example

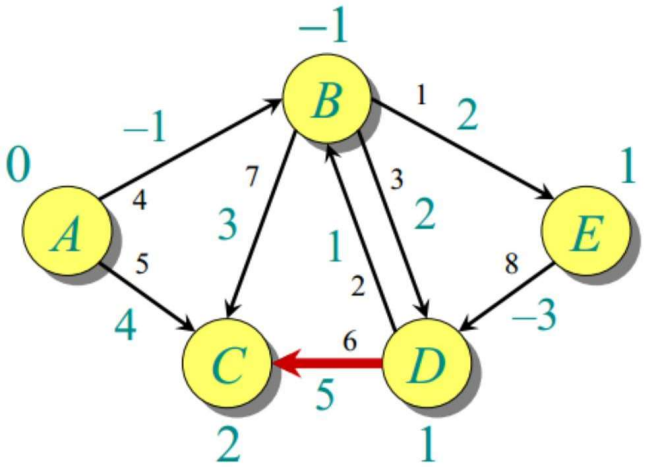


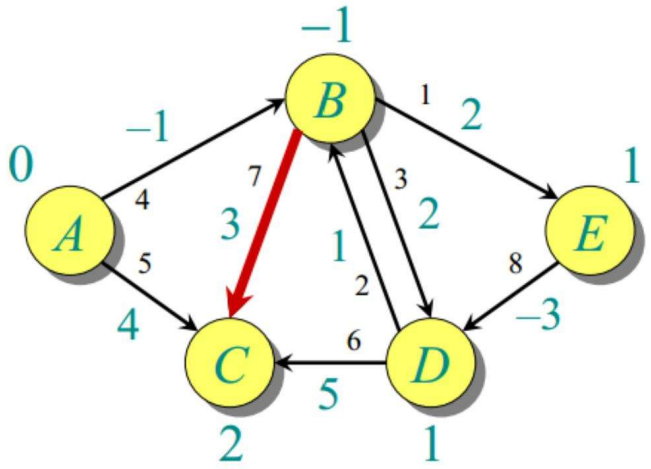


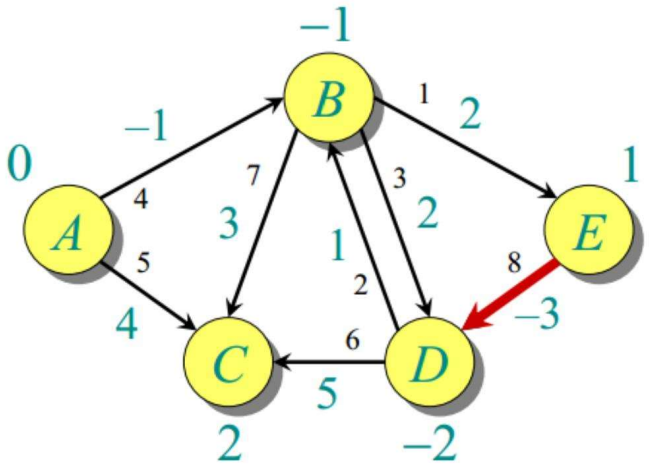


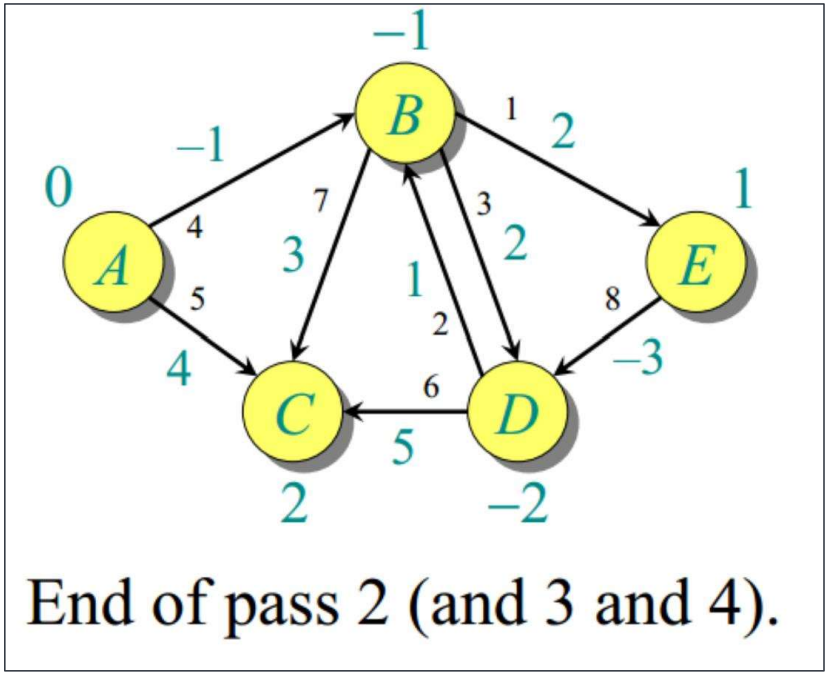






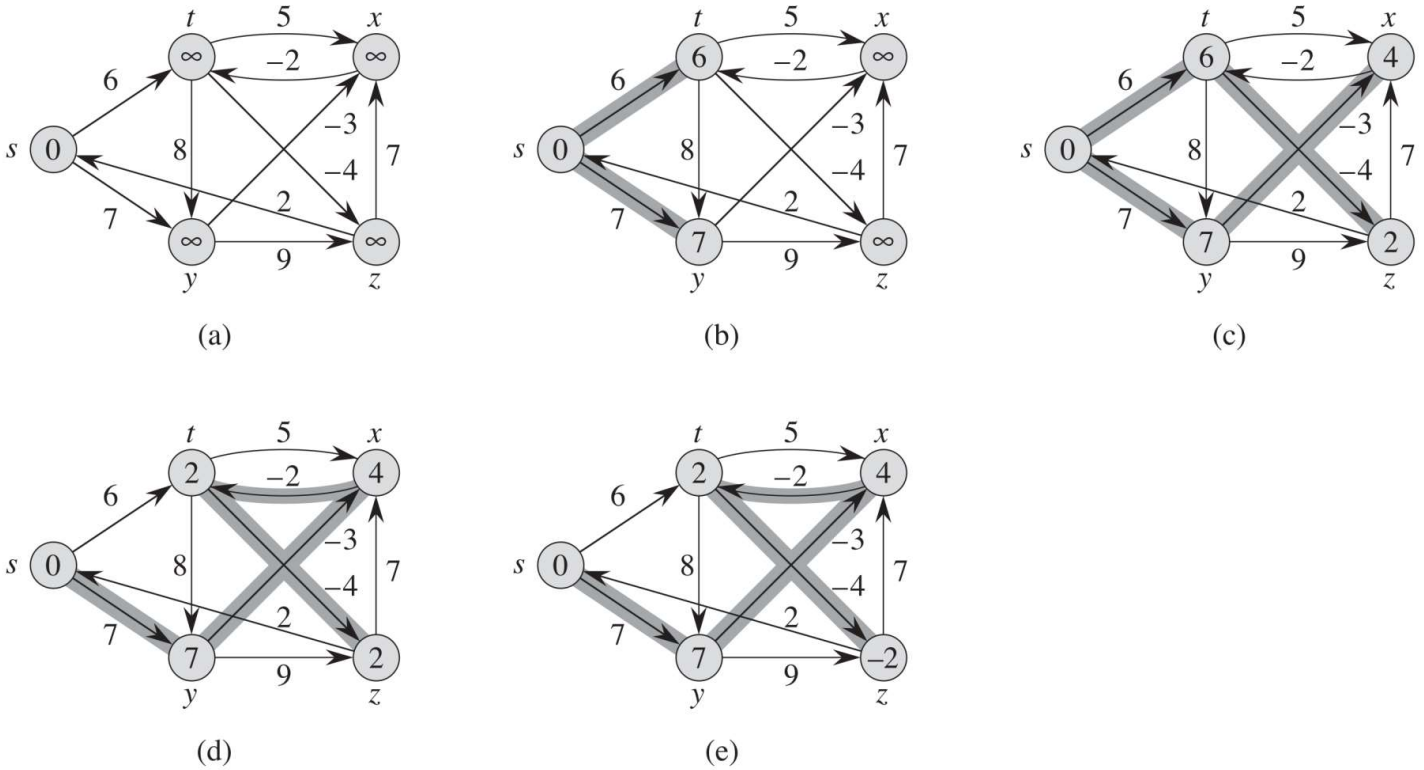






End of pass 2 (and 3 and 4).

Bellman-Ford Algorithm: Another Example



Bellman-Ford Algorithm

- Prove: after |V|-1 passes, all *d* values correct
  - Consider shortest path from s to v:

    $s \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v$
    - Initially, d[s] = 0 is correct, and doesn't change
    - After 1 pass through edges, $d[v_1]$ is correct and doesn't change
    - After 2 passes, $d[v_2]$ is correct and doesn't change
    - …
    - Terminates in |V| - 1 passes: (*Why?*)

Contents of this presentation are also based on
Chapter-24, Introduction to Algorithms (3rd edition) Cormen, Leiserson, Rivest, & Stein

Additional Reference
https://www.cs.utexas.edu/~tandy/barrera
MIT 6.046J/18.401J Introduction to Algorithms (Lecture 18 by Prof. Erik Demaine)

STEVENS
INSTITUTE OF TECHNOLOGY
1870

THANK **YOU**

Stevens Institute of Technology