Lecture 8: Graph Algorithms – II

Tsung-Wei (TW) Huang

Department of Electrical and Computer Engineering

University of Utah, Salt Lake City, UT

Programming Assignment #1

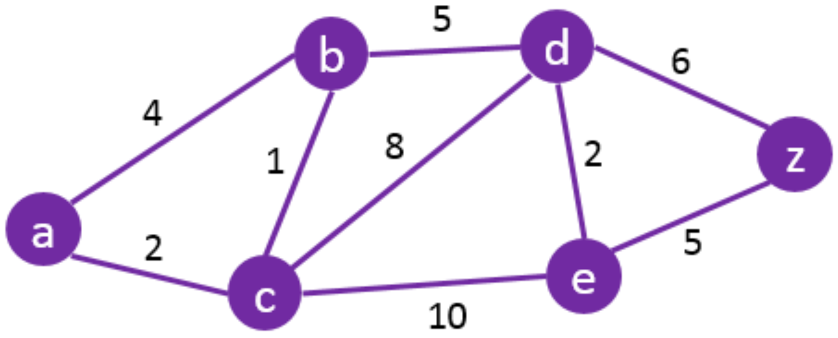
- **Implement FM partitioning algorithm**
  - https://github.com/tsung-wei-huang/ece5960-physical-design/tree/main/PA1
- **Two checkpoint dues, 9/7 and 9/14 23:59 PM**
  - https://github.com/tsung-wei-huang/ece5960-physical-design/issues/2
- **Final Due on 9/21 (Wed) 23:59 PM**
  - Upload your solutions to twhuang-server-01.ece.utah.edu
    - Account: ece6960-fall22
  - Place your source code + README under PA1/your_uid/
  - README should contain instruction to compile & run your code

Programming Assignment #1 (cont'd)

- **In addition to source code + README, upload a report with:**
  - A table showing your results of each benchmark
  - A section discussing what challenges you encounter
  - A section discussing how you overcome those challenges
    - Also discuss unsolved challenges
- **The report needs to be just a one- or two-page pdf**
  - No need to be lengthy …
- **Upload your report to the class GitHub page**
  - https://github.com/tsung-wei-huang/ece5960-physical-design/issues/1
  - **Due 9/21 23:59 PM (today!!!)**

Shortest Path Algorithm

- **Shortest path finding problem formulation**
  - Given a *weighted graph G*, find the minimum-weight path from a given source vertex s to all other vertices
- **Tremendous applications**
  - Map: what is the shortest path from SLC to CA?
  - Circuit design: what is the minimum interconnect?
  - GPS: what is the shortest path to rescue the car?



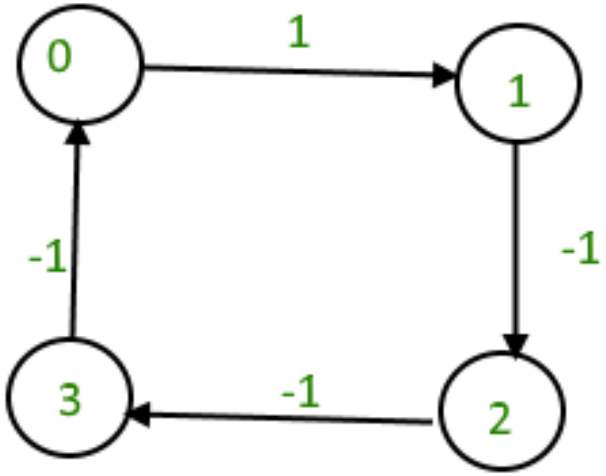
What is the shortest path from A to Z?

A→C→B→D→E→Z: cost = 15

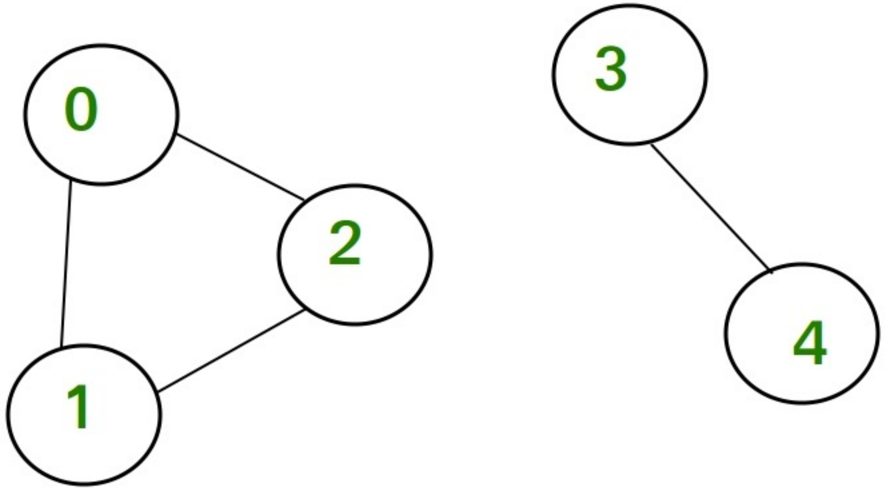
A→C→D→Z: cost = 16

Shortest Path May Not Exist

- If graph contains non-reachable targets
- If graph contains negative cycles



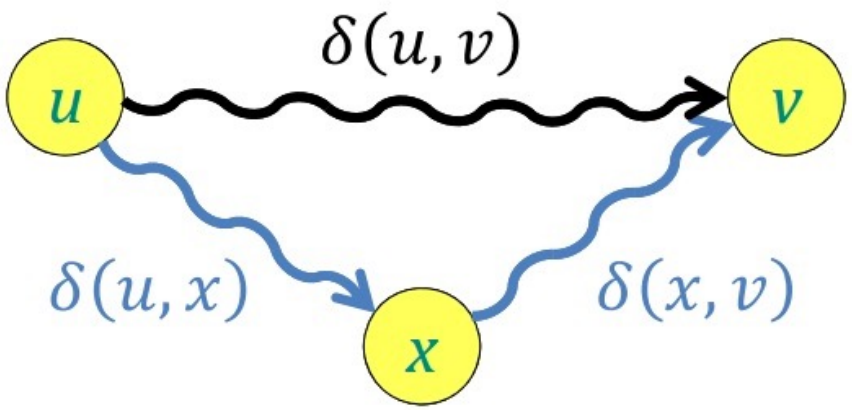
Negative cycle



No route from vertex 0 to vertex 4

Shortest Path Property

- ***Optimal substructure***
  - The shortest path consists of shortest subpaths
  - Easy to prove by contradiction
- **Let $\delta(u,v)$ be the the shortest path from u to v**
  - Shortest paths satisfy the *triangle inequality*
    - $\delta(u,v) \leq \delta(u,x) + \delta(x,v)$

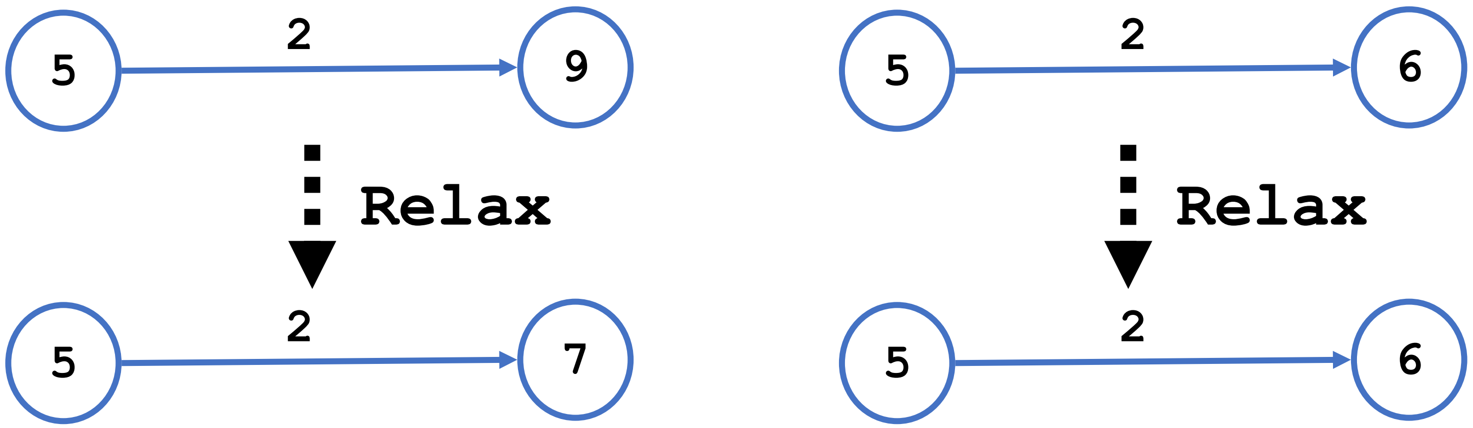


Key idea: With the optimal substructure property, we can apply a greedy iterative algorithm to improve the distance from the source to all other nodes! (remembered we have done this in KL-FM partition)

Relaxation

- Key technique: *relaxation*
  - Maintain upper bound d[*v*] on δ(s,v):

```
Relax(u,v,w) {
   if (d[v] > d[u]+w) then d[v]=d[u]+w;
}
```



Bellman-Ford Shortest Path Algorithm

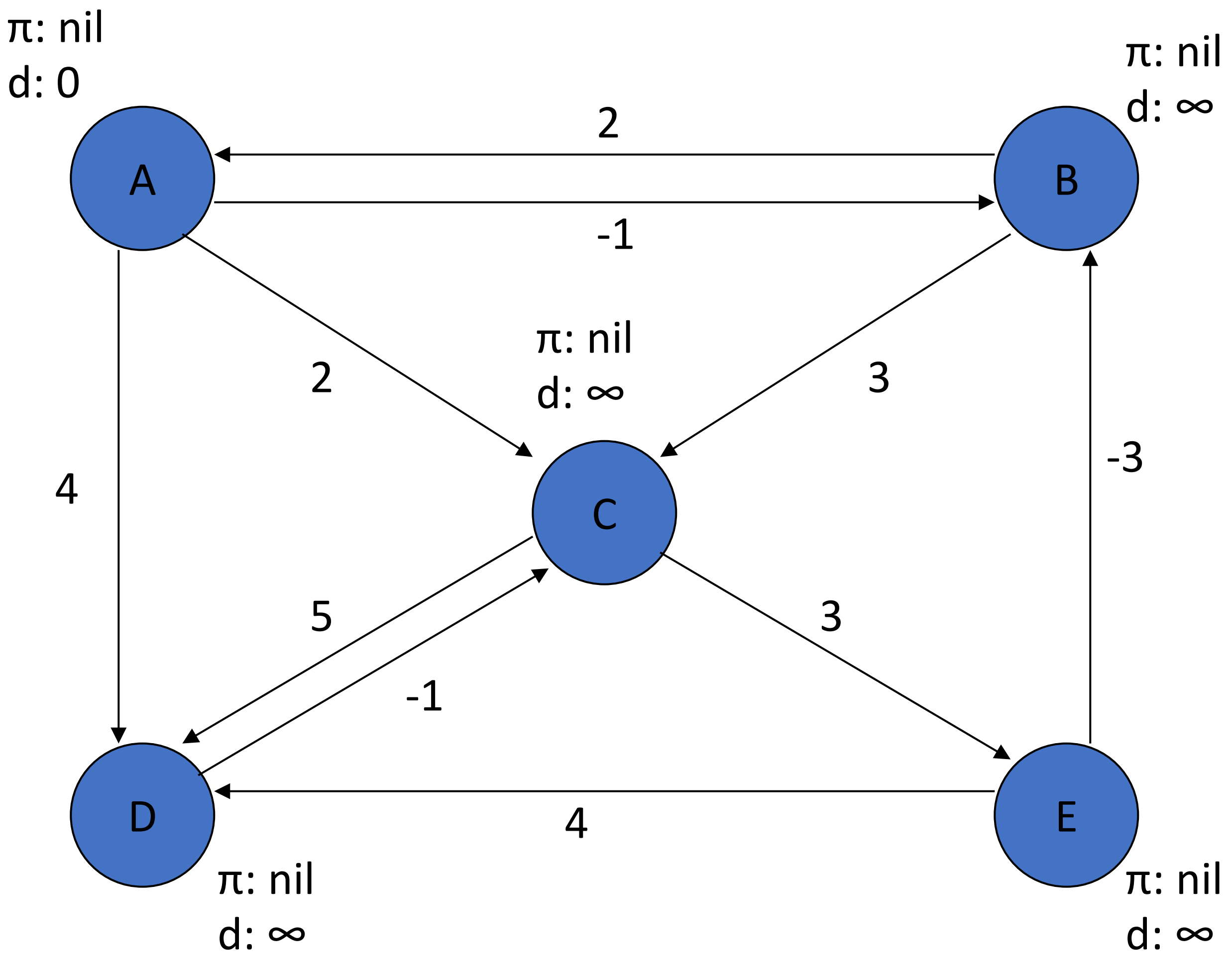
```
BellmanFord()
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0;
   for i=1 to |V|-1
      for each edge (u,v) ∈ E
         Relax(u,v, w(u,v));
```

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

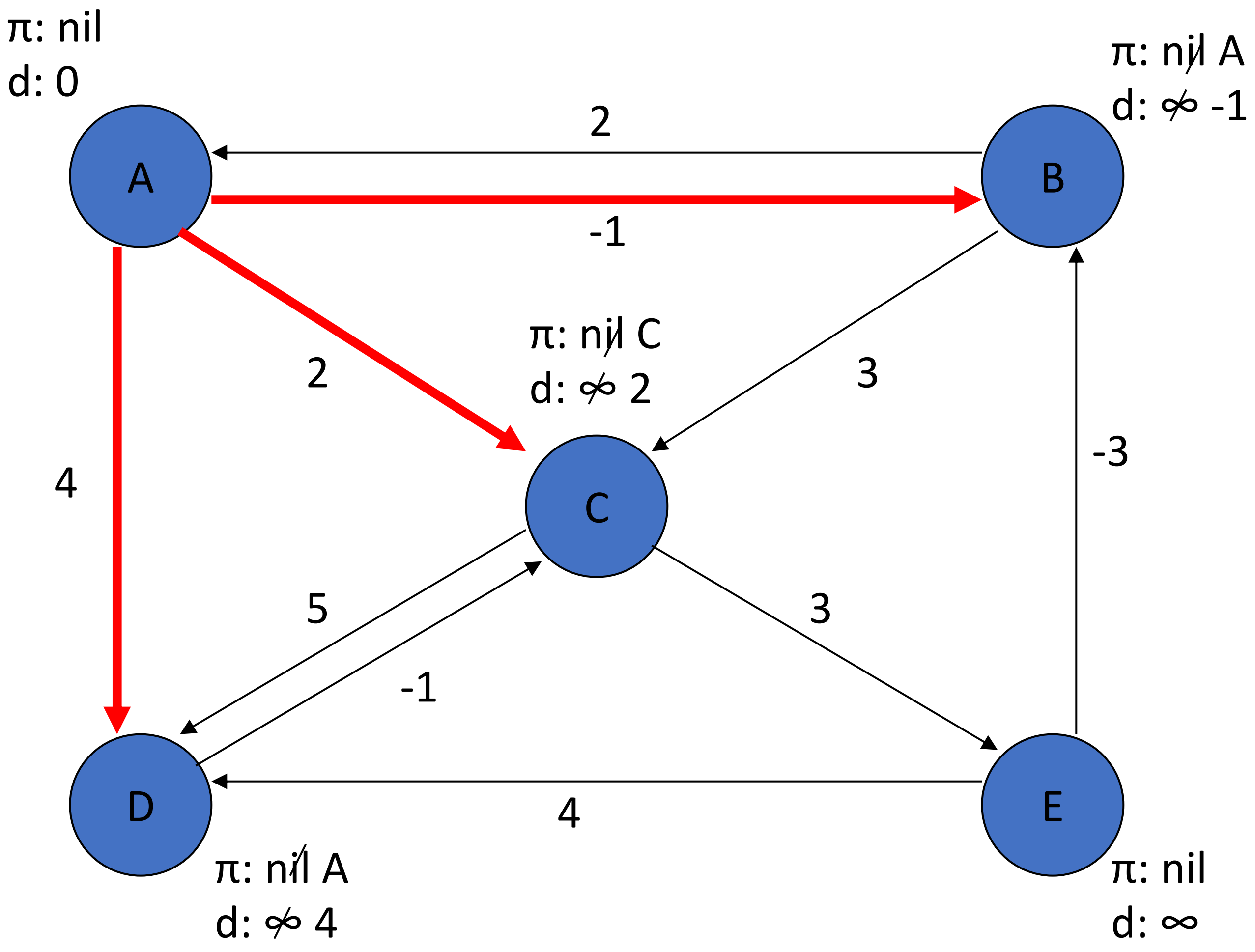
Relaxation: Make |V|-1 passes, relax each edge if possible

```
Relax(u,v,w): if (d[v] > d[u]+w) then d[v]=d[u]+w
```

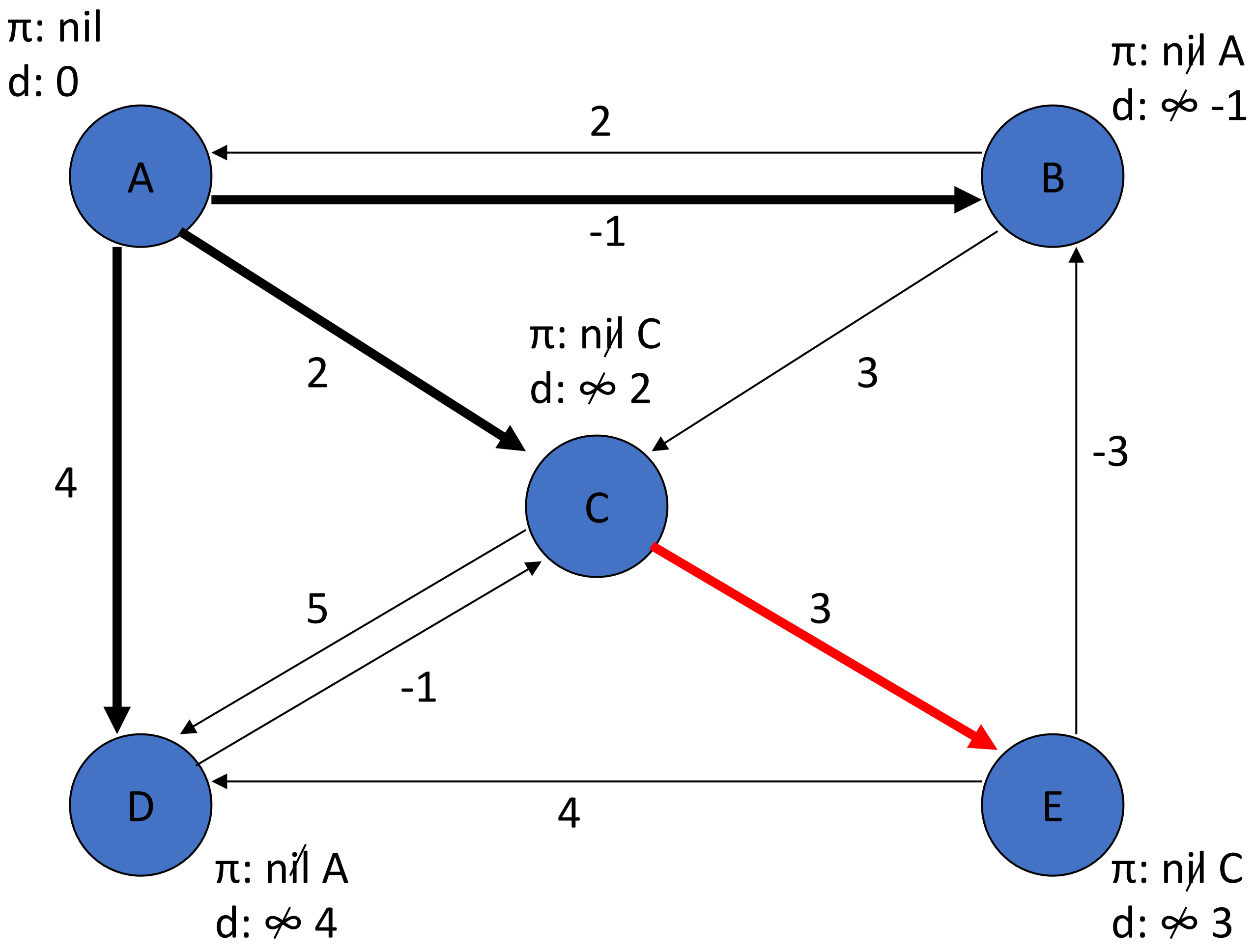
Bellman-Ford Algorithm Walkthrough – 1



Bellman-Ford Algorithm Walkthrough – 2



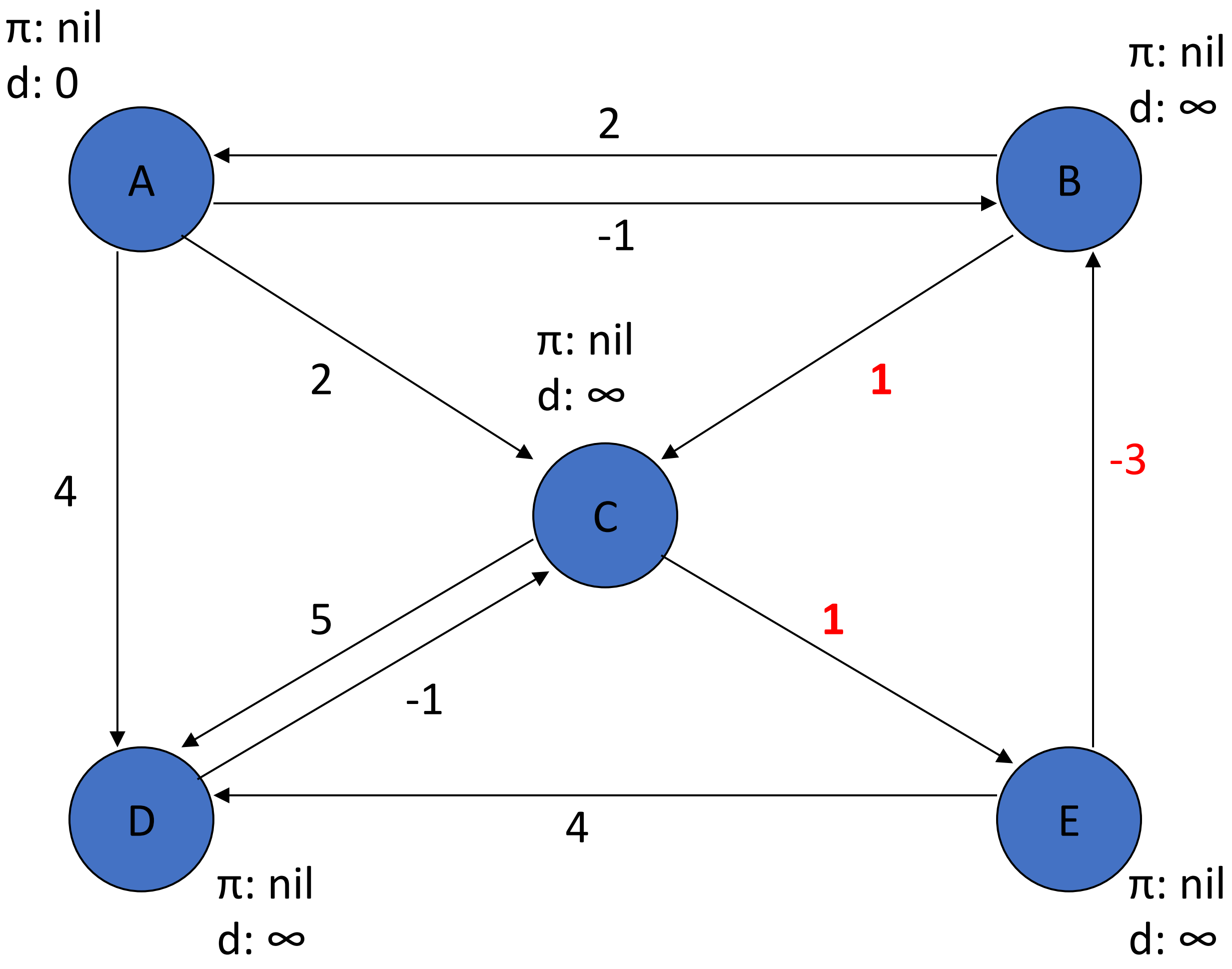
Bellman-Ford Algorithm Walkthrough – 3



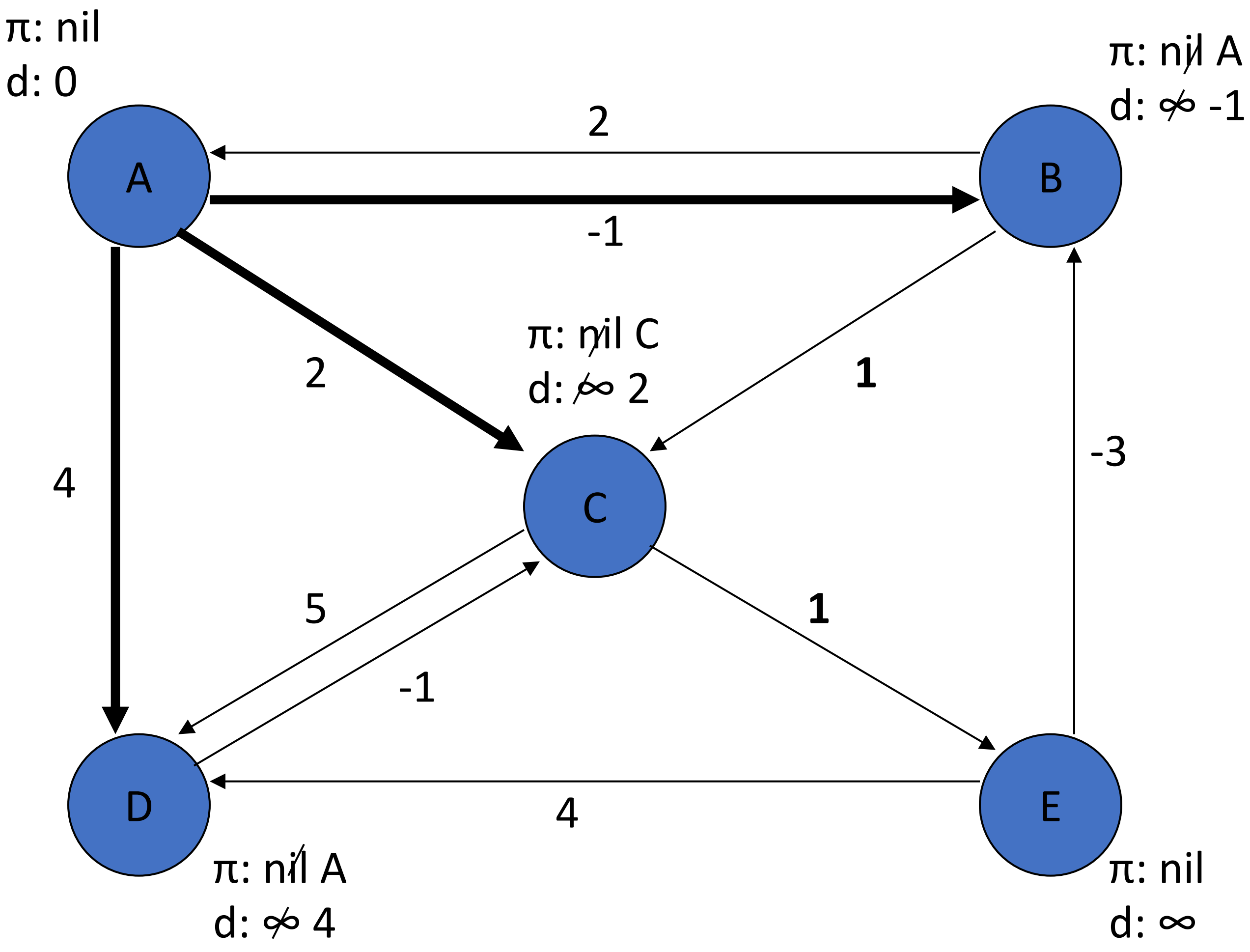
Algorithm Complexity

- **Running time: O(VE)**
  - Not so good for large dense graphs
  - But a very practical algorithm in many ways
- **What about graph with negative cycles?**
  - For the case that shortest path does not exist

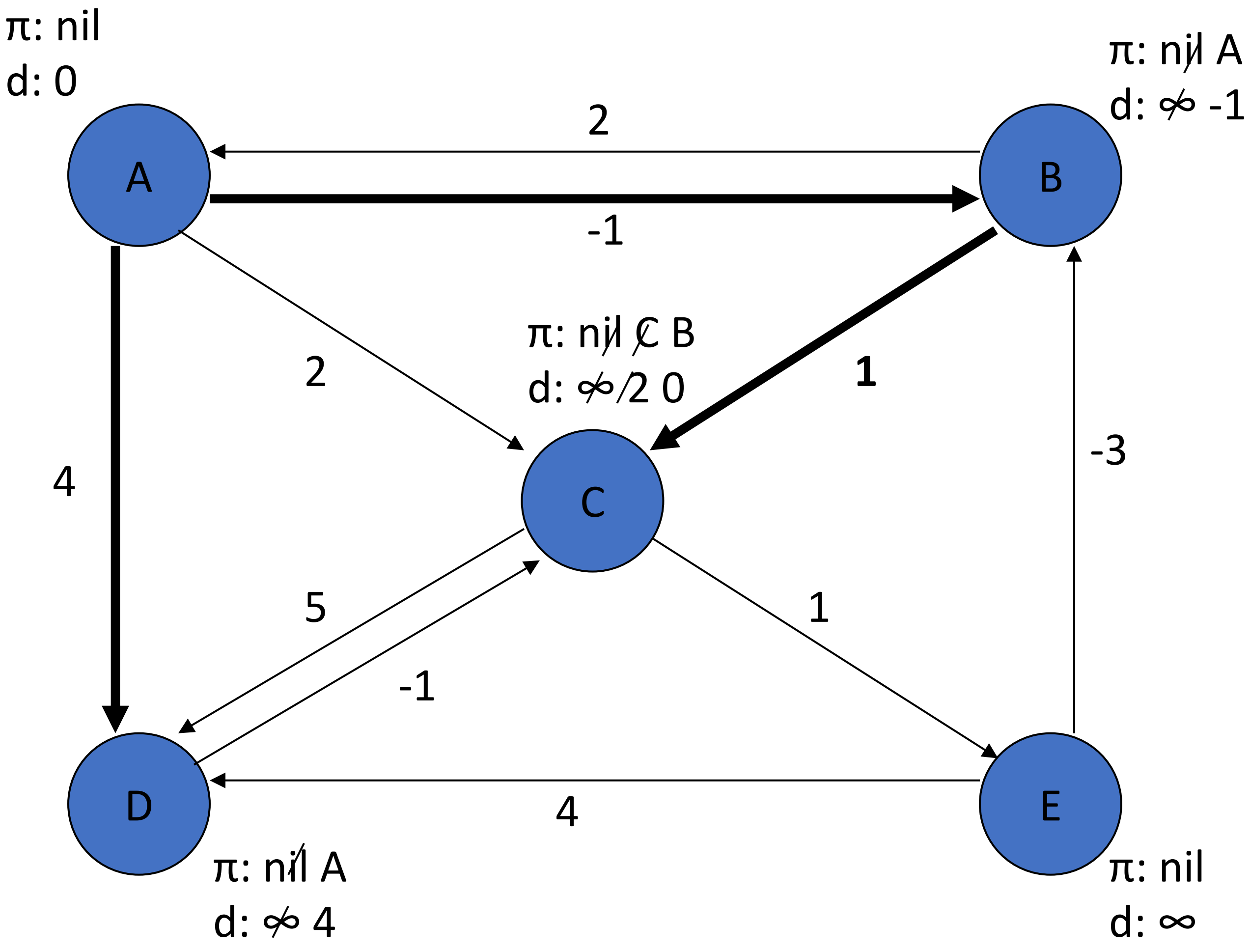
Negative Cycle Walkthrough – 1



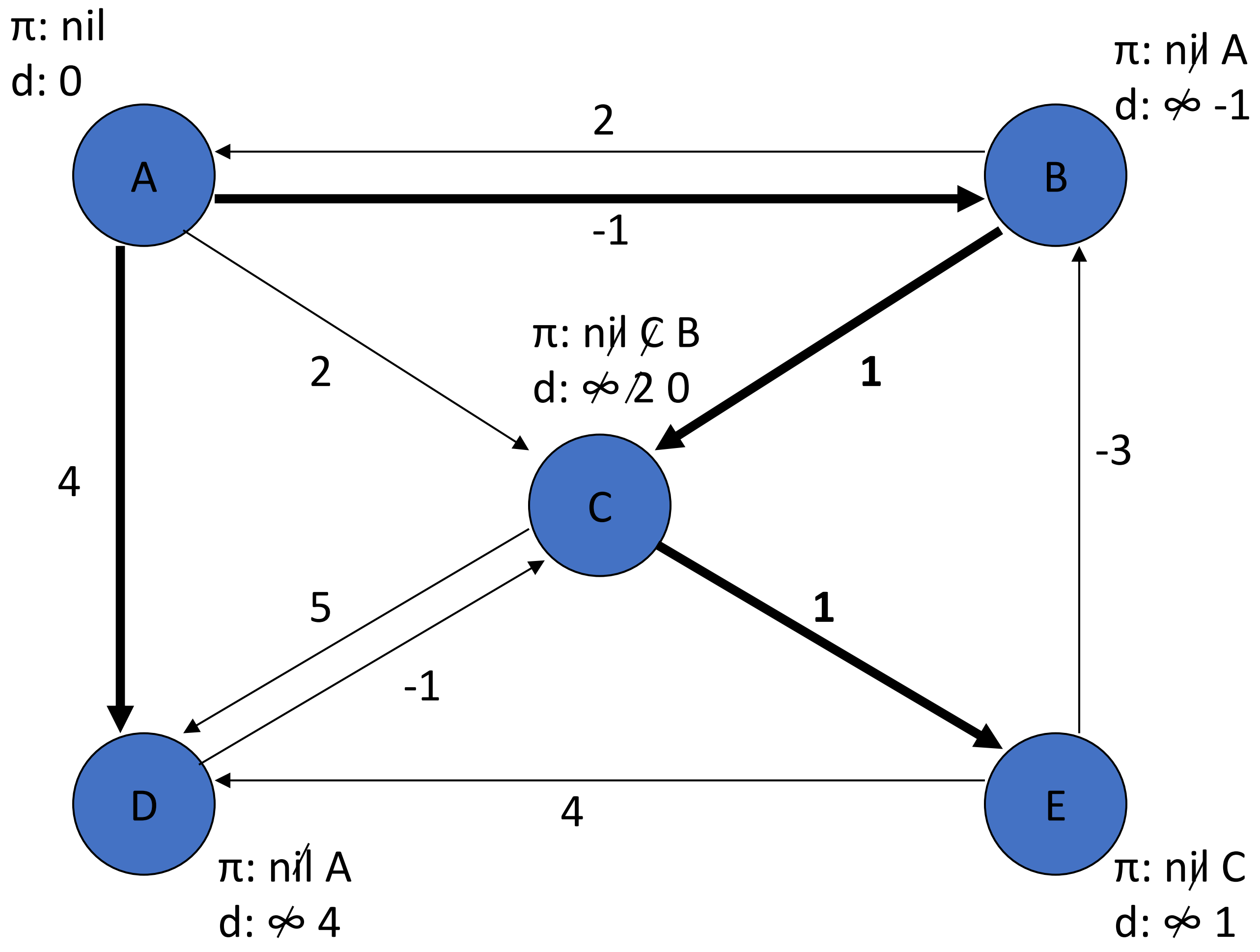
Negative Cycle Walkthrough – 2



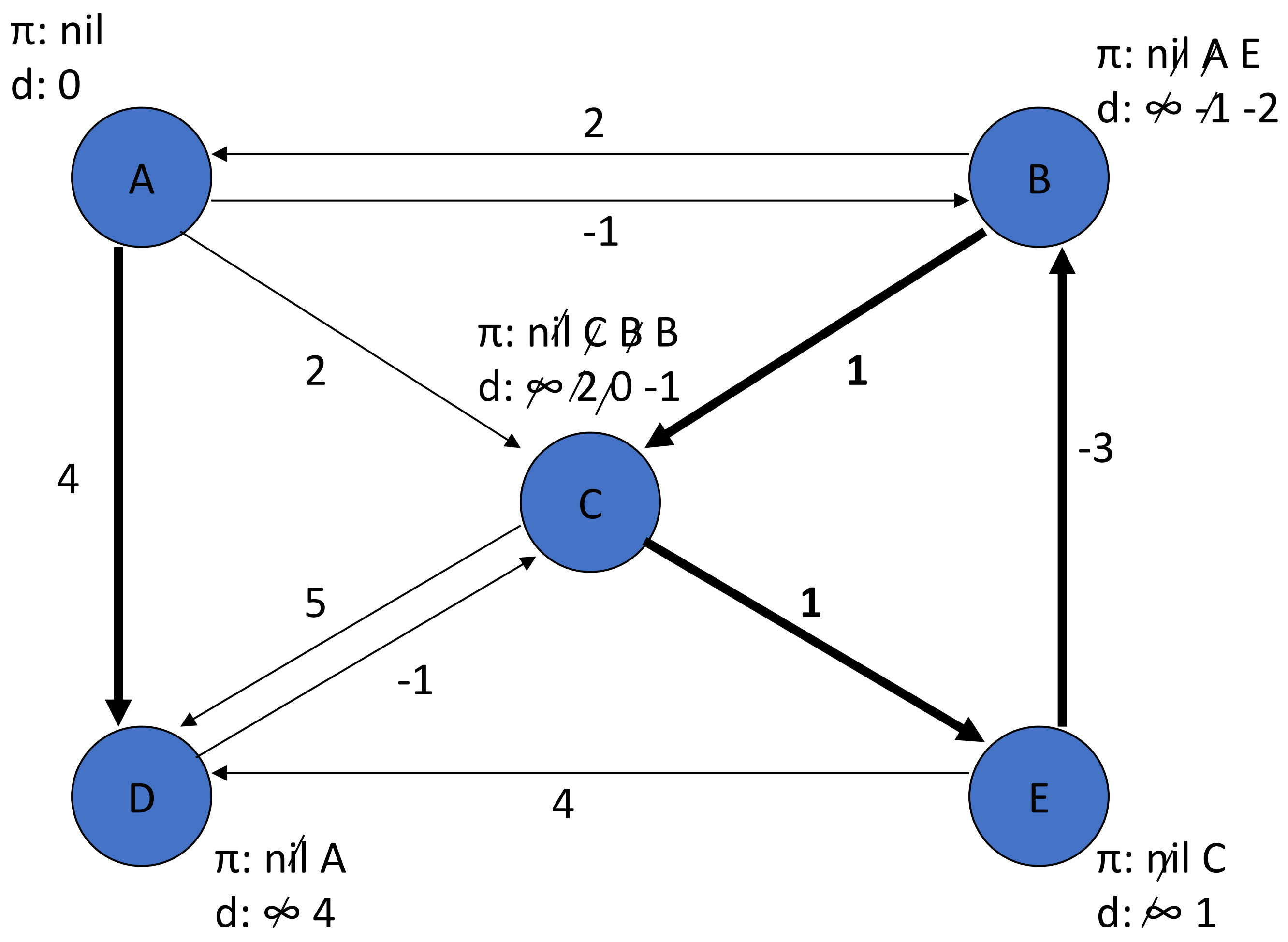
Negative Cycle Walkthrough – 3



Negative Cycle Walkthrough – 4



Negative Cycle Walkthrough – 5



Negative Cycle Detection

```
BellmanFord()
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0;
   for i=1 to |V|-1
      for each edge (u,v) ∈ E
         Relax(u,v, w(u,v));
   for each edge (u,v) ∈ E
      if (d[v] > d[u] + w(u,v))
           return "no solution";
```

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

Relaxation: Make |V|-1 passes, relax each edge if possible

Negative cycle test: have we converged yet? i.e., no more relaxations are possible after V passes

```
Relax(u,v,w): if (d[v] > d[u]+w) then d[v]=d[u]+w
```

Algorithm Complexity (cont'd)

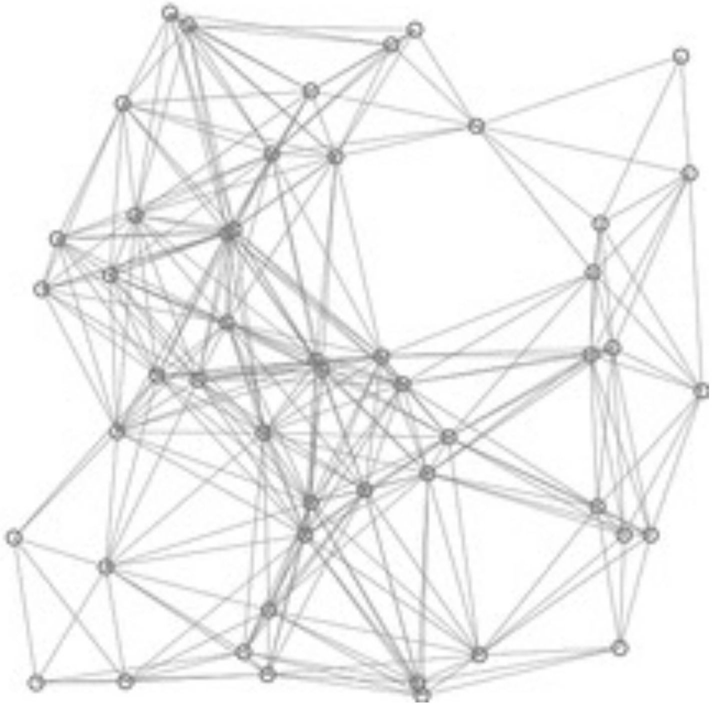
- **Running time: O(VE)**
  - Not so good for large dense graphs
  - But a very practical algorithm in many ways
- **Pros and Cons?**

Improvement of Bellman-Ford Algorithm

- **Shortest-path-faster algorithm (SPFA)**
  - Only perform relaxation from active vertices
  - Largely reduce the relaxation times

```
procedure SPFA(G, s)
for each vertex v ≠ s in V(G)
    d(v) := ∞
d(s) := 0
push s into Q
while Q is not empty do
    u := Q.pop()
    for each edge (u, v) in E(G) do
        if d(u) + w(u, v) < d(v) then
            d(v) := d(u) + w(u, v)
            if v is not in Q then
                push v into Q
```

Relaxation happens only at active vertices (in queue) – *largely reduced redundant relaxations!*



red lines are shortest path covering
blue lines are relaxations

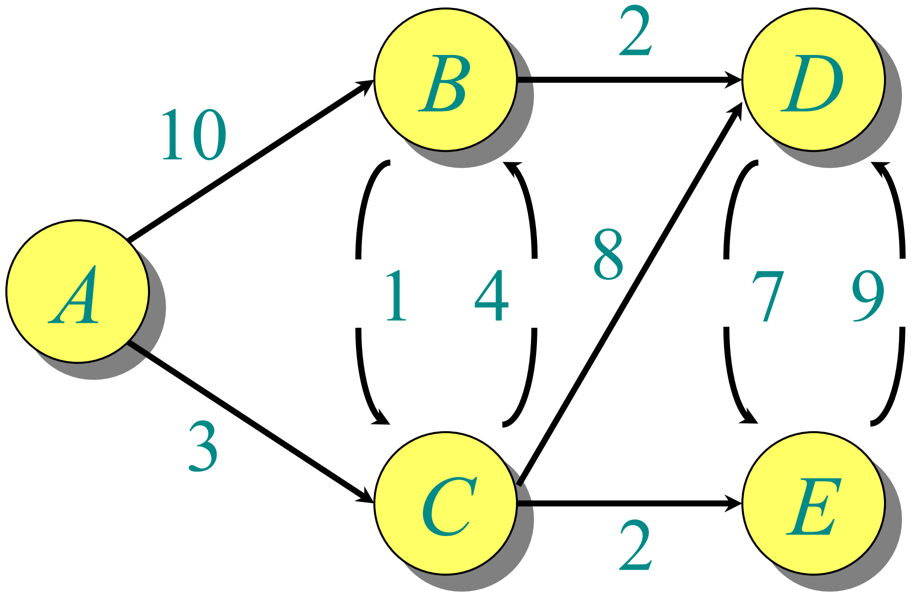
From SPFA to Dijkstra Algorithm

d[*s*] ← 0
for each *v* Î *V* – {*s*}
    do *d*[*v*] ← ¥
S ← ∅
Q ← *V*   ⊳ *Q* Replacing the SPFA queue with min priority queue!
while *Q* ¹ ∅
    do *u* ← EXTRACT-MIN(*Q*)
        S ← *S* È {*u*}
        for each *v* Î *Adj*[*u*]
            do if *d*[*v*] > *d*[*u*] + *w*(*u*, *v*)
                then *d*[*v*] ← *d*[*u*] + *w*(*u*, *v*)
                    p[*v*] ← *u*

Relaxation happens only at the active vertex with the minimum (shortest) value discovered so far! (i.e., vertex *u*)

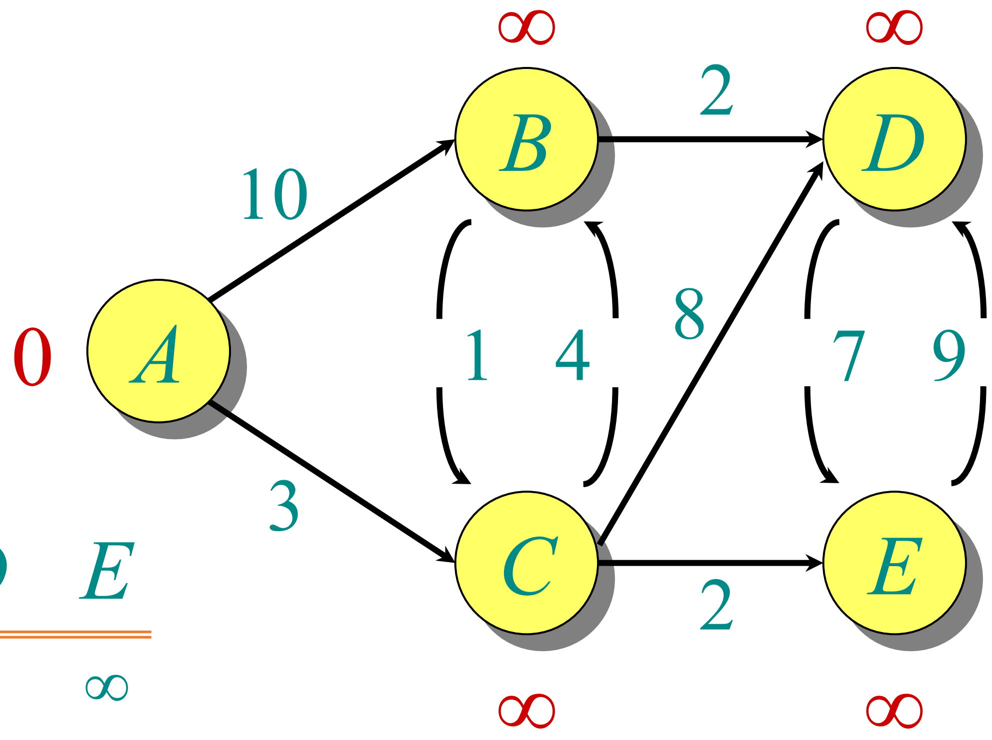
Dijkstra Algorithm Walkthrough – 1

Graph with <u>nonnegative</u> edge weights



Dijkstra Algorithm Walkthrough – 2

Initialize:

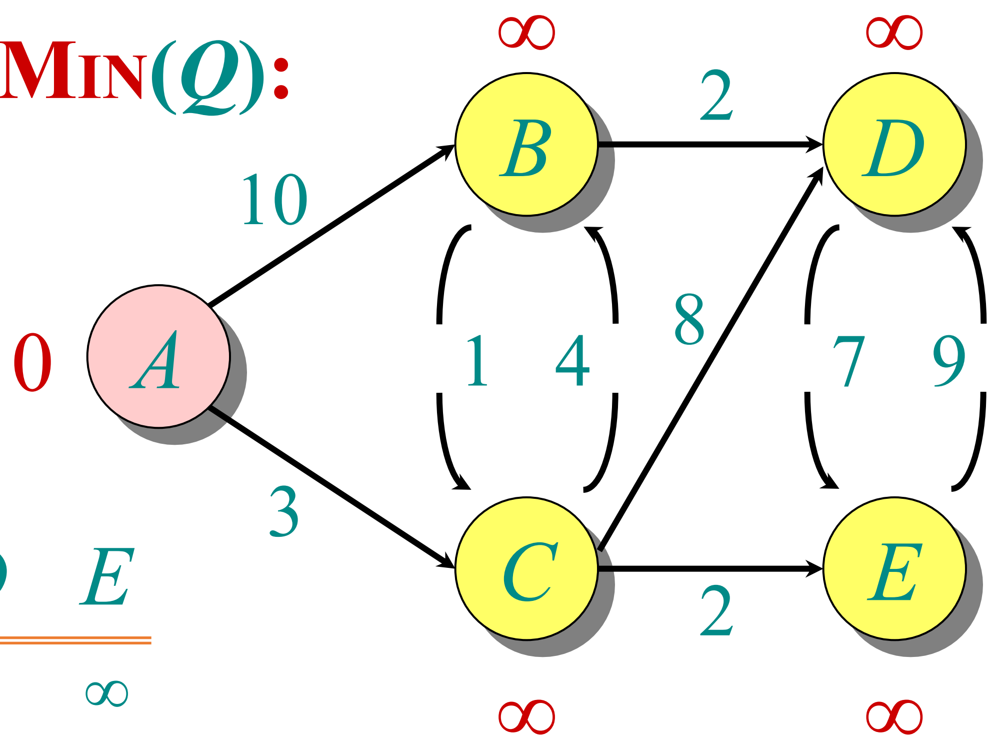


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: {}

Dijkstra Algorithm Walkthrough – 3

"A" ← **EXTRACT-MIN**(Q):

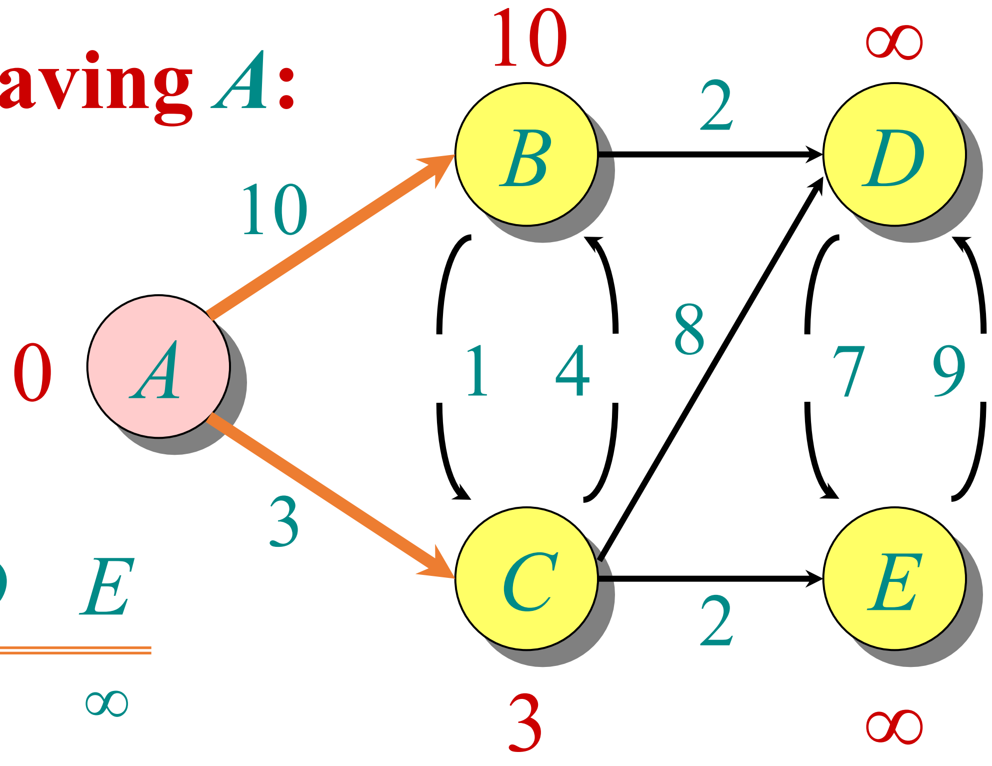


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: { A }

Dijkstra Algorithm Walkthrough – 4

Relax all edges leaving *A*:

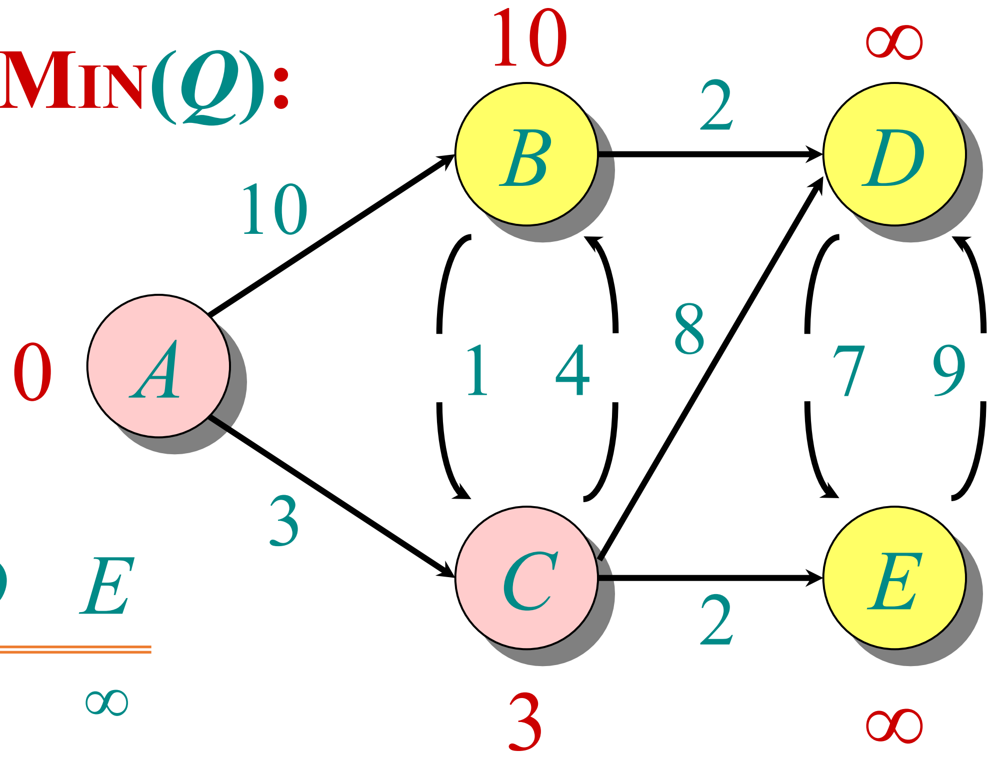


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

Dijkstra Algorithm Walkthrough – 5

"C" ← EXTRACT-MIN(Q):

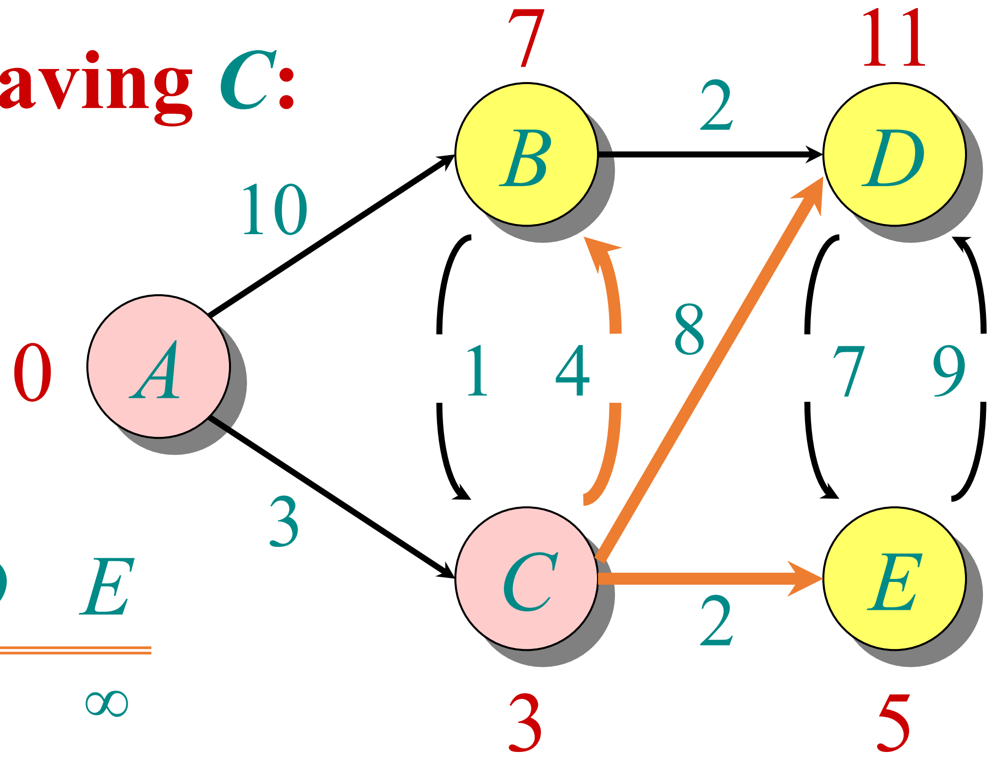


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A, C }

Dijkstra Algorithm Walkthrough – 6

Relax all edges leaving *C*:

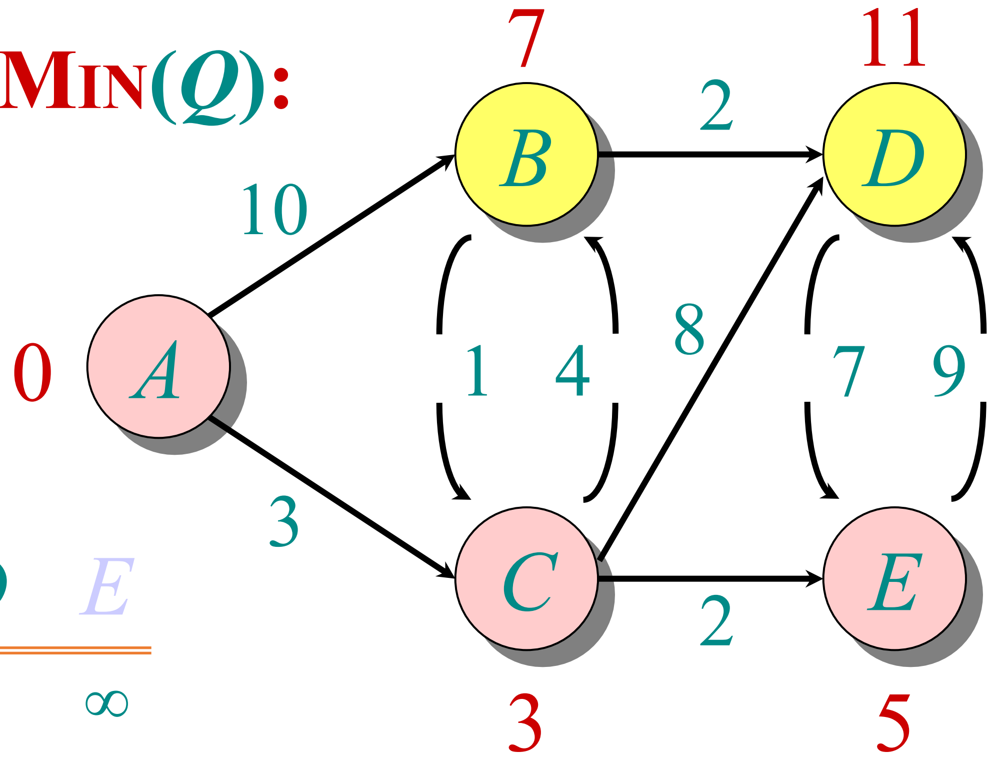


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C }

Dijkstra Algorithm Walkthrough – 7

"E" ← EXTRACT-MIN(Q):

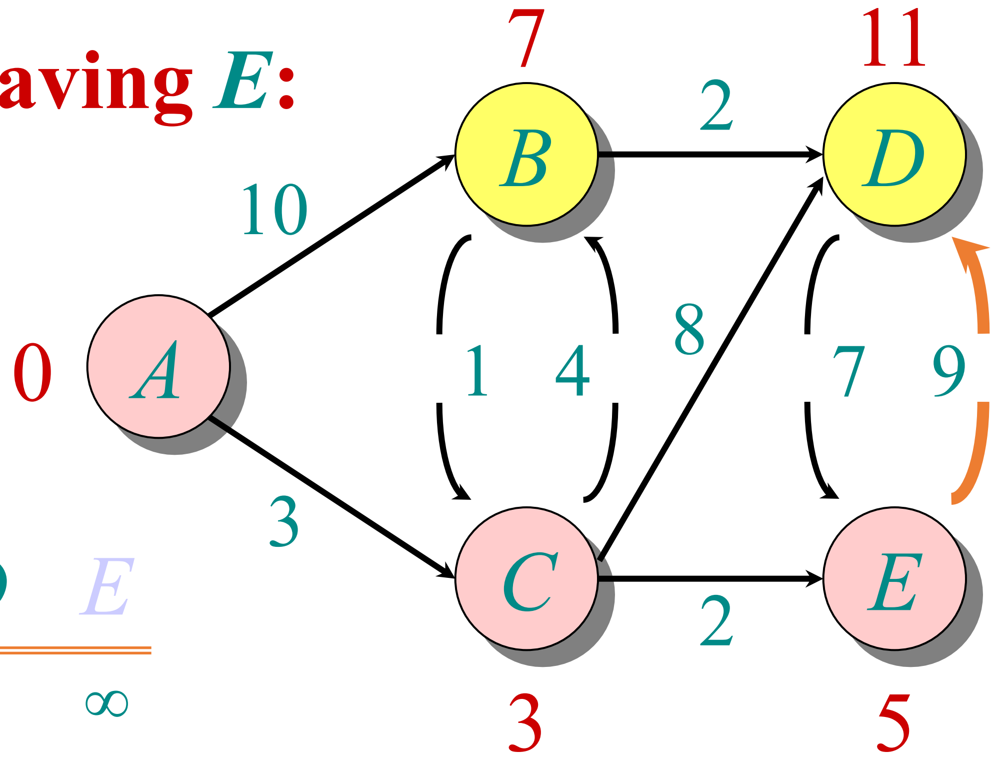


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C, E }

Dijkstra Algorithm Walkthrough – 8

Relax all edges leaving *E*:

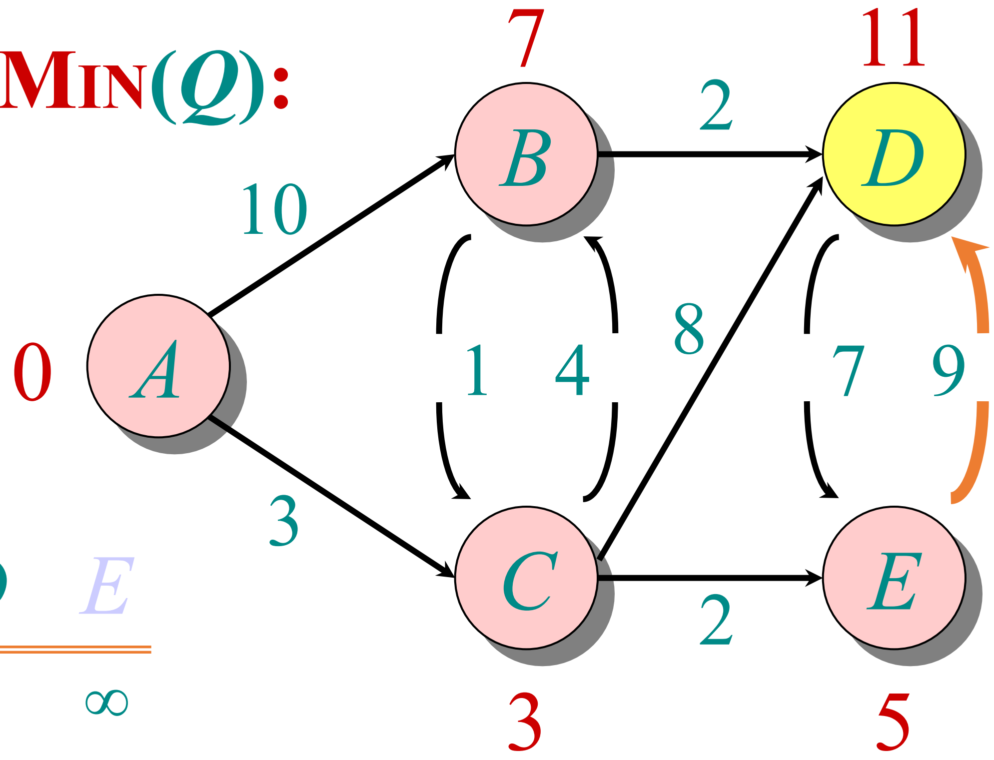


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E }

Dijkstra Algorithm Walkthrough – 9

"*B*" ← EXTRACT-MIN(Q):

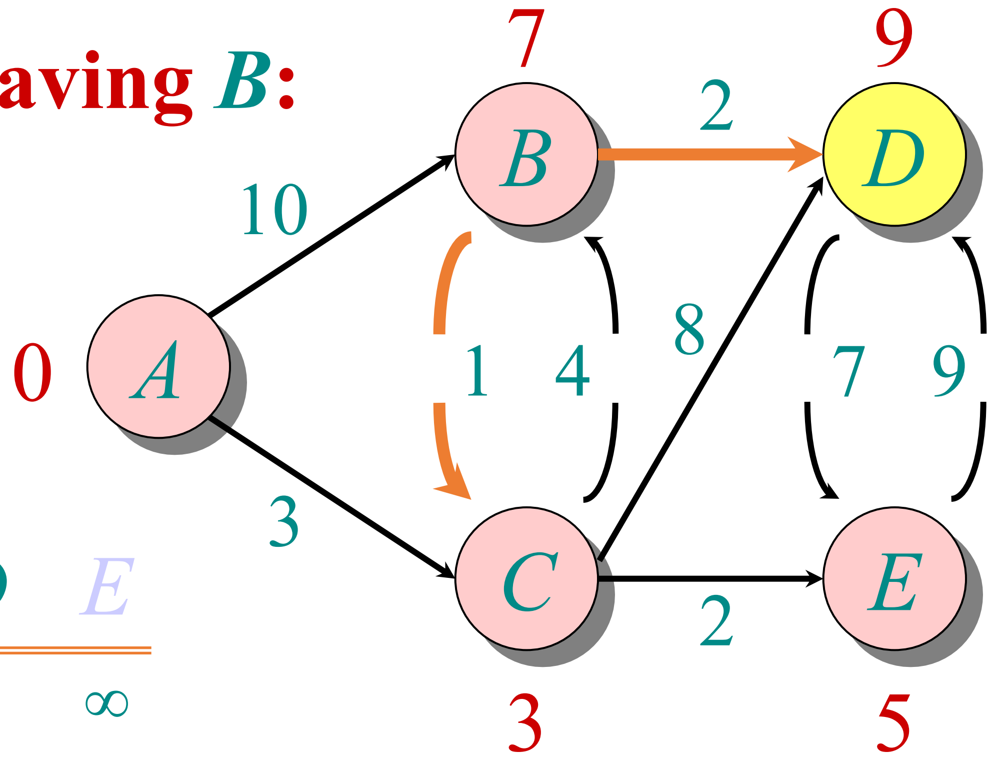


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E, B }

Dijkstra Algorithm Walkthrough – 10

Relax all edges leaving *B*:



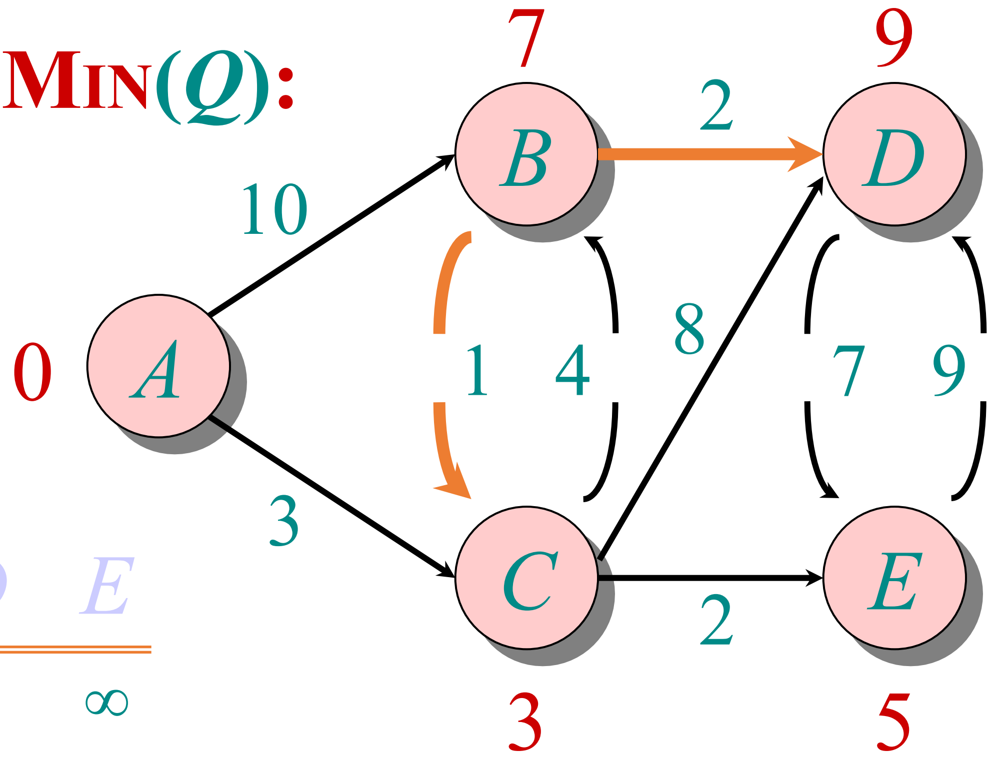
Q:

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

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

Dijkstra Algorithm Walkthrough – 11

"*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* }

Dijkstra Algorithm Complexity

- **Running time: $O(|V|\log|V| + |E|)$**
  - Priority queue operations are all $\log|V|$
  - $|V|$ times of pops and insertions
  - $|E|$ relaxations
- **Pros and Cons?**

Summary

- **We have discussed three shortest path algorithms**
  - Bellman-Ford algorithm
  - Shortest-path-faster algorithm (SPFA)
  - Dijkstra algorithm
- **In practice, SPFA works very well in many graphs**
  - Easy to implement
  - Can detect negative cycles as well
- **Dijkstra has the lowest complexity but**
  - Requires priority queue (more complicated than the normal queue)
  - Requires graphs to have non-negative weights