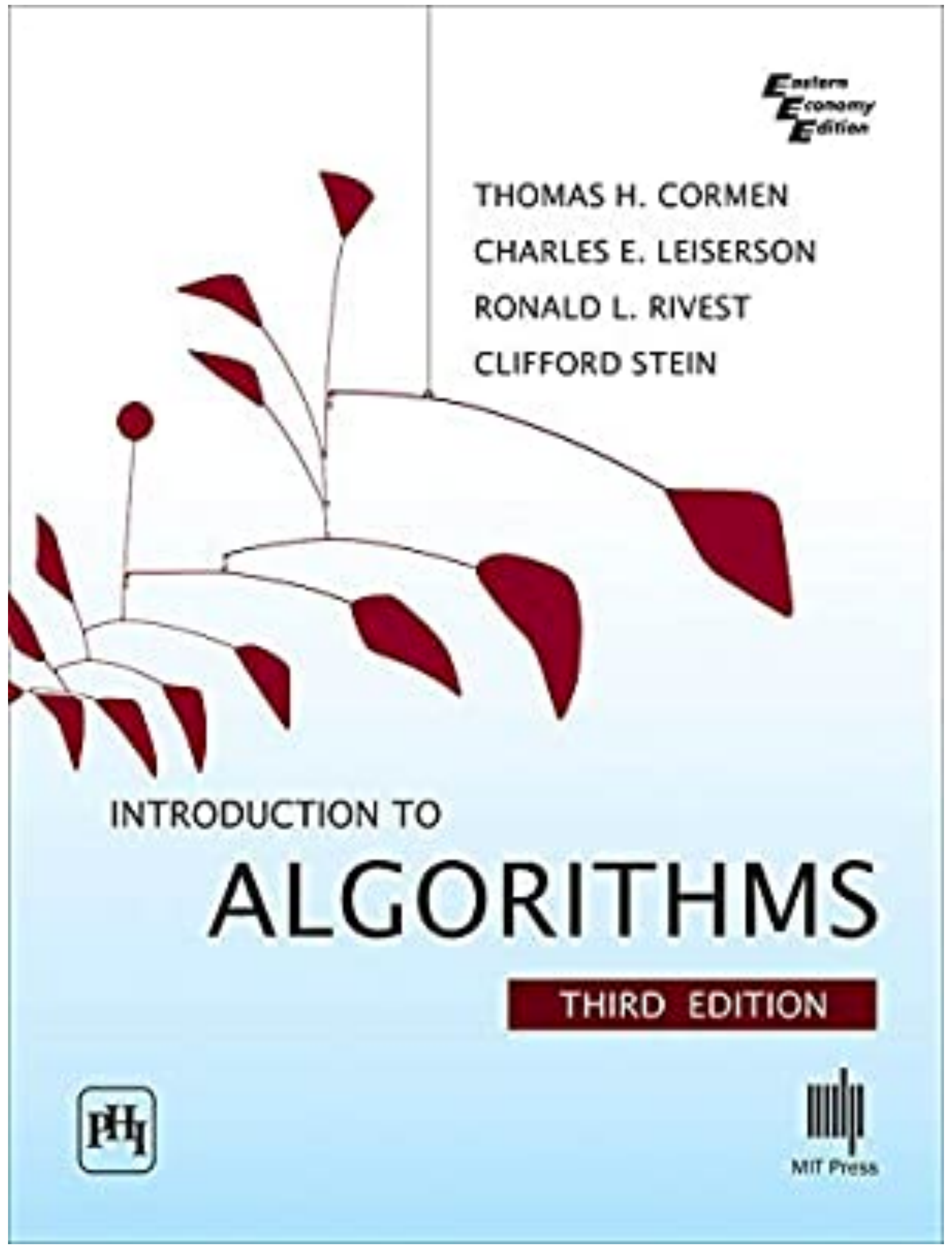
Lecture 4: Shortest Paths II - Dijkstra

Overview

- Review
- Shortest paths in DAGs
- Shortest paths in graphs without negative edges
- Dijkstra’s Algorithm

Readings



CLRS, Sections 24.2-24.3

Shortest path weight

Notation:

$v_0 \overset{p}{\longrightarrow} v_k$ means p is a path from v_0 to v_k. (v_0) is a path from v_0 to v_0 of weight 0.

Definition:

Shortest path weight from u to v as

$$\delta(u,v) = \begin{cases} \min\left\{ w(p) : u \overset{p}{\longrightarrow} v \right\} & \text{if } \exists \text{ any such path} \\ \infty & \text{otherwise} \quad (v \text{ unreachable from } u) \end{cases}$$

General structure of S.P. Algorithms

Initialize: for $v \in V$: $\begin{array}{ccc} d[v] & \leftarrow & \infty \\ \Pi[v] & \leftarrow & \text{NIL} \end{array}$

$d[S] \leftarrow 0$

Main: repeat

select edge (u, v) [somehow]

"Relax" edge (u, v)

$\left[\begin{array}{l} \text{if } d[v] > d[u] + w(u, v) : \\ \quad d[v] \leftarrow d[u] + w(u, v) \\ \quad \pi[v] \leftarrow u \end{array}\right.$

until all edges have $d[v] \leq d[u] + w(u, v)$

Relaxation is Safe

Basic operation in shortest path computation is the relaxation operation:

$$\begin{array}{l} \mathrm{RELAX}(u, v, w) \\ \qquad \text{if } d[v] > d[u] + w(u, v) \\ \qquad\quad \text{then } d[v] \leftarrow d[u] + w(u, v) \\ \qquad\qquad\quad \Pi[v] \leftarrow u \end{array}$$

Lemma: The relaxation algorithm maintains the invariant that $d[v] \geq \delta(s, v)$ for all $v \in V$.

Proof: By induction on the number of steps. Consider RELAX(u, v, w). By induction $d[u] \geq \delta(s, u)$. By the triangle inequality, $\delta(s, v) \leq \delta(s, u) + \delta(u, v)$. This means that $\delta(s, v) \leq d[u] + w(u, v)$, since $d[u] \geq \delta(s, u)$ and $w(u, v) \geq \delta(u, v)$. So setting $d[v] = d[u] + w(u, v)$ is safe.

DAGs

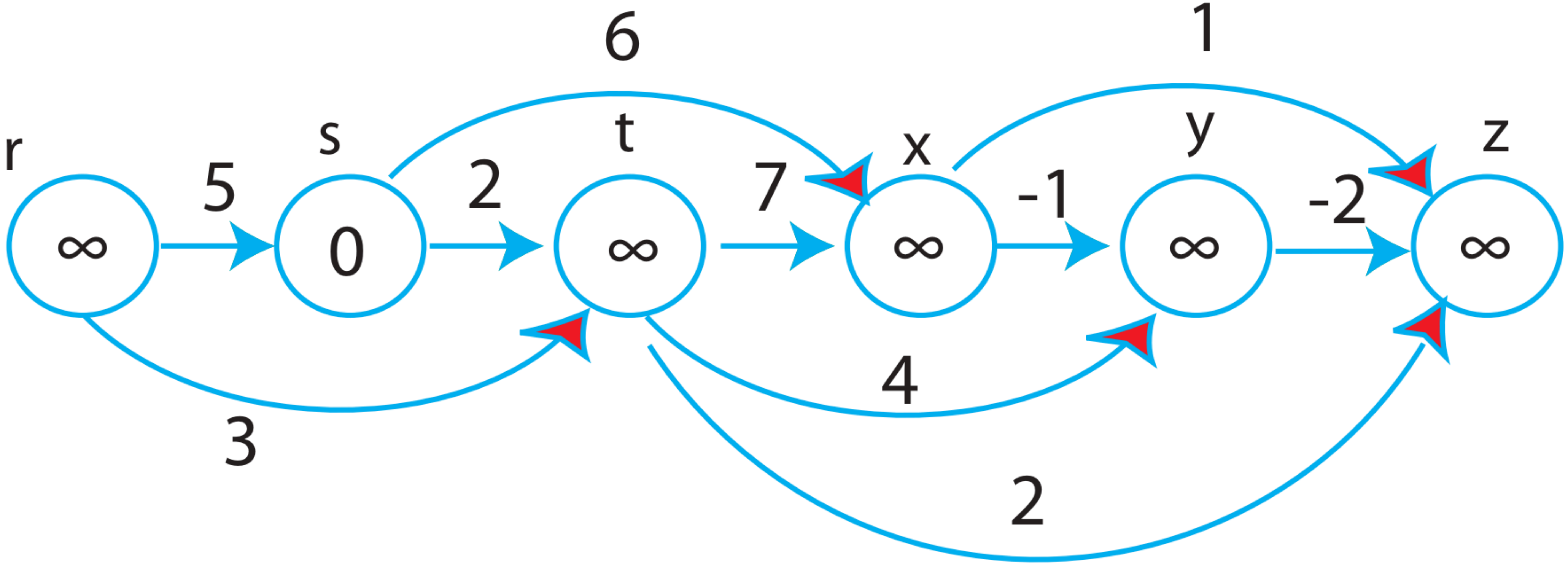
Can't have negative cycles because there are no cycles!

1. Topologically sort the DAG. Path from u to v implies that u is before v in the linear ordering.

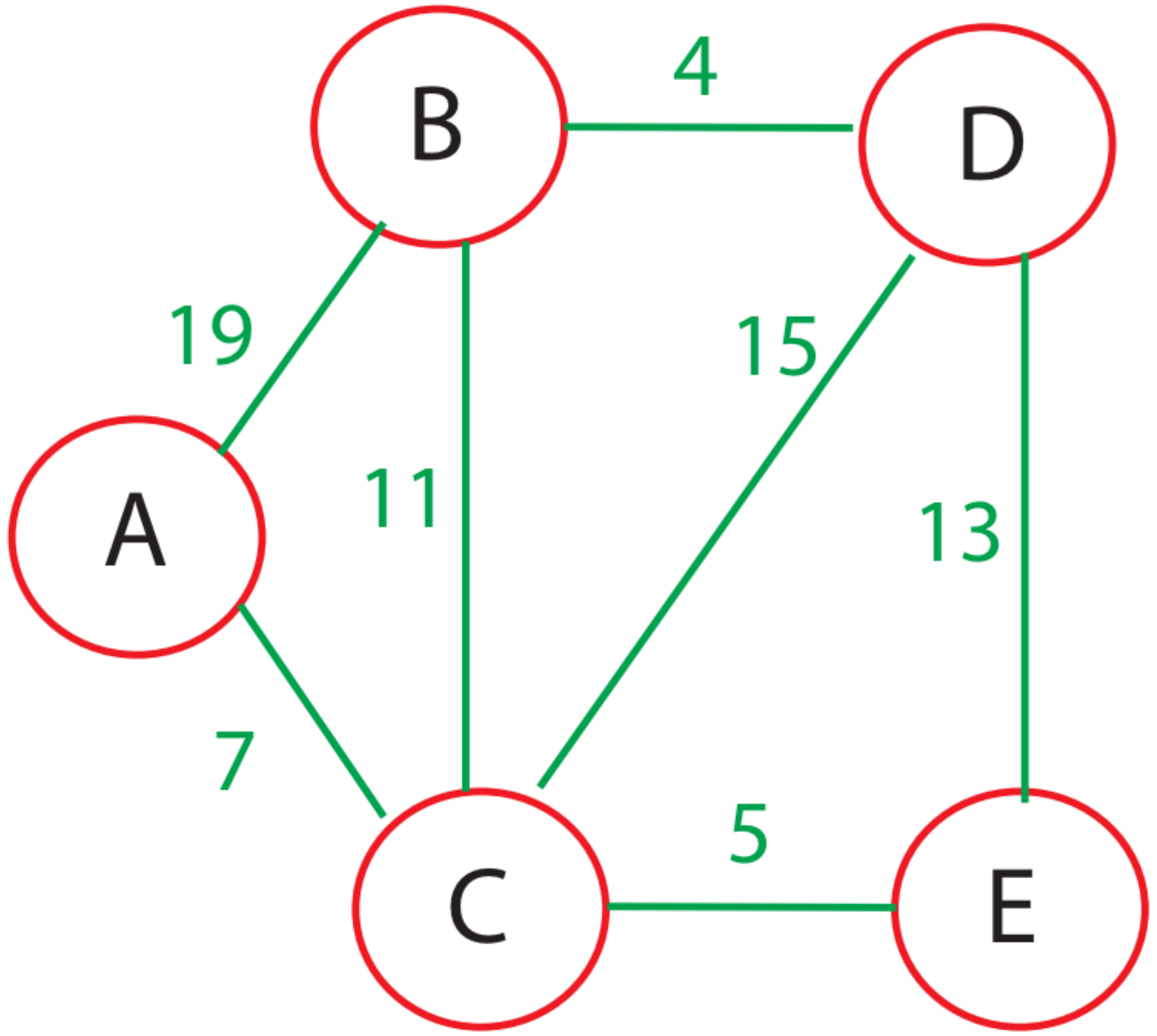
2. One pass over vertices in topologically sorted order relaxing each edge that leaves each vertex.

$\Theta(V + E)$ time

Example DAG



Dijkstra Example:



A	C	E	B	D
	7	12	18	22

D	B	E	C	A
	4	13	15	22

E	C	A	D	B
	5	12	13	16

Dijkstra's Algorithm

For each edge $(u, v) \epsilon E$, assume $w(u, v) \geq 0$, maintain a set S of vertices whose final shortest path weights have been determined. Repeatedly select $u \epsilon V - S$ with <u>minimum</u> shortest path estimate, add u to S, relax all edges out of u.

Pseudo-code

```
Dijkstra (G, W, s)        //uses priority queue Q
        Initialize (G, s)
        S ← ϕ
        Q ← V[G]        //Insert into Q
        while Q ≠ ϕ
              do u ← EXTRACT-MIN(Q)        //deletes u from Q
              S = S ∪ {u}
              for each vertex v ϵ Adj[u]
                      do RELAX (u, v, w)    ← this is an implicit DECREASE_KEY operation
```

Example

Dijkstra (G, W, s) //uses priority queue Q

  Initialize (G, s)

  $S \leftarrow \phi$

  $Q \leftarrow V[G]$ //Insert into Q

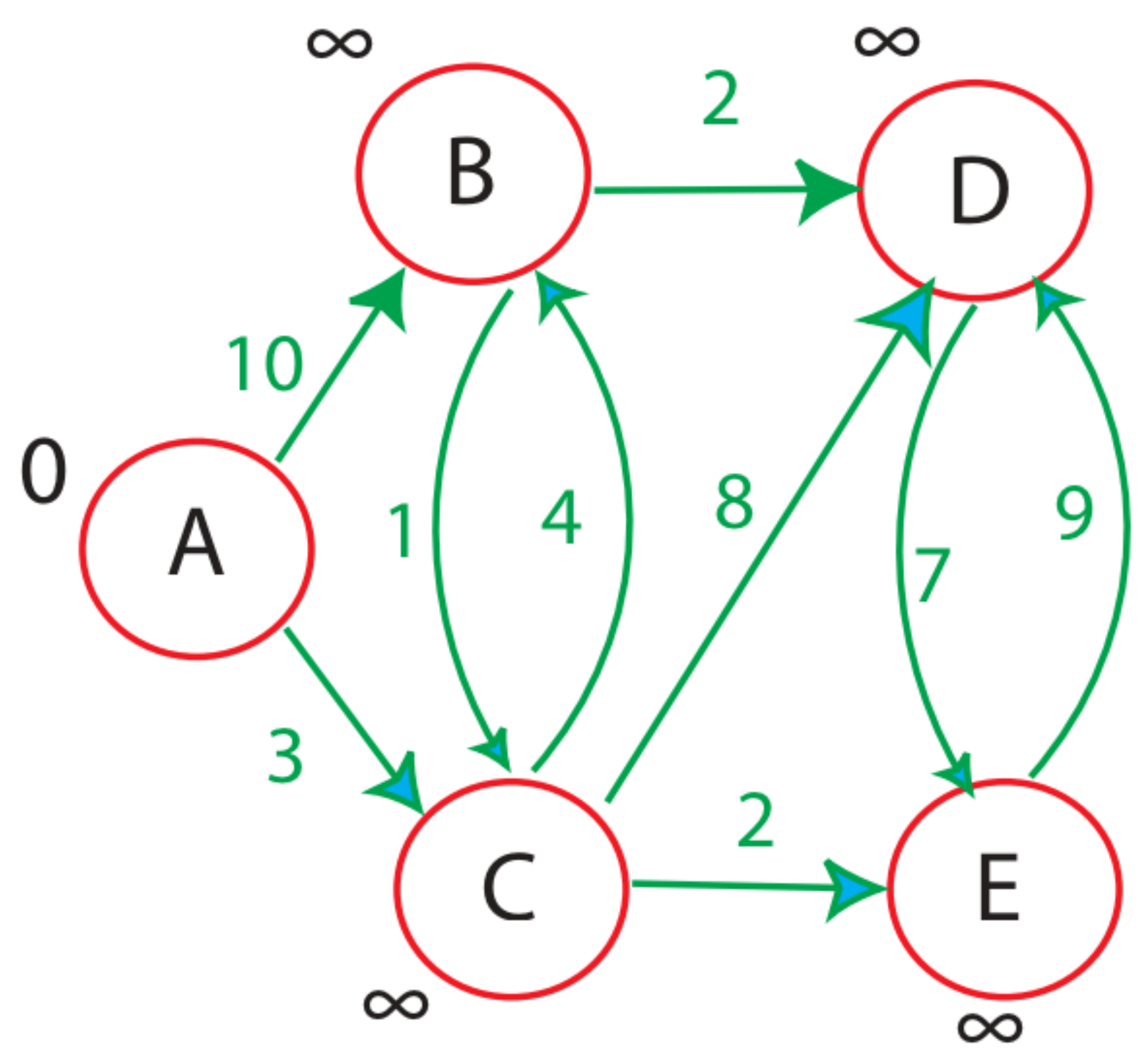
  while $Q \neq \phi$

    do $u \leftarrow$ EXTRACT-MIN(Q)

    $S = S \cup \{u\}$

    for each vertex $v \in$ Adj$[u]$

      do RELAX (u, v, w)



S = { }	{ A	B	C	D	E }	= Q
S = { A }	0	∞	∞	∞	∞	
S = { A, C }	0	10	3	∞	∞	← after relaxing edges from A
S = { A, C }	0	7	3	11	5	← after relaxing edges from C
S = { A, C, E }	0	7	3	11	5	
S = { A, C , E, B}	0	7	3	9	5	← after relaxing edges from B

Dijkstra Algorithm Summary

***Strategy**: Dijkstra is a greedy algorithm: choose closest vertex in V – S to add to set S.*

***Correctness**: We know relaxation is safe. The key observation is that each time a vertex u is added to set S, we have d[u] = δ(s, u).*

Dijkstra Complexity
Θ(v) inserts into priority queue
Θ(v) EXTRACT MIN operations
Θ(E) DECREASE KEY operations

Dijkstra Algorithm Summary

Dijkstra Complexity
Θ(v) inserts into priority queue
Θ(v) EXTRACT MIN operations
Θ(E) DECREASE KEY operations

Array impl:

$\Theta(v)$ time for extra min
$\Theta(1)$ for decrease key
Total: $\Theta(V.V + E.1) = \Theta(V^2 + E) = \Theta(V^2)$

Binary min-heap:

$\Theta(\lg V)$ for extract min
$\Theta(\lg V)$ for decrease key
Total: $\Theta(V \lg V + E \lg V)$