Chapter 24
Single-source Shortest Paths

The slides for this course are based on the course textbook: Cormen, Leiserson, Rivest, and Stein, Introduction to Algorithms, 3rd edition, The MIT Press, McGraw-Hill, 2010.

Chapter 24 Topics

- What are single-source shortest paths?
- Dijkstra’s algorithm for finding a shortest path

Shortest Path

$G = (V, E)$ weighted directed graph

$w: E \rightarrow \Re$ weight function

Path $p = <v_0, v_1, \ldots, v_n>$

Weight of a path $w(p) = \sum_{i=1}^{n} w(v_{i-1} v_i)$

Shortest Path

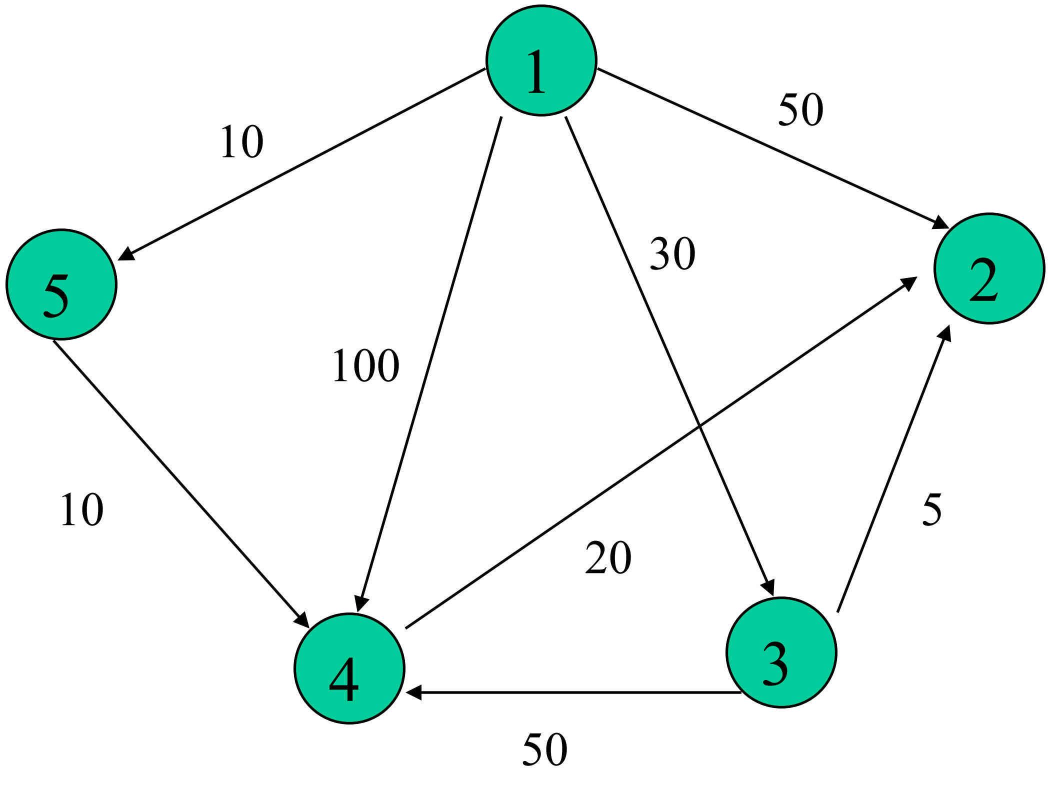
Shortest path weight from u to v:

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\mapsto} v\} & \text{if a } u,v \text{ path exits} \\ \infty & \text{otherwise} \end{cases}$$

Shortest path from u to v: Any path from u to v with w(p) = $\delta(u,v)$

$\pi[v]$ Predecessor of v on a path

Shortest path



Variants

- Single-source shortest paths:
  - find shortest paths from source vertex to every other vertex
- Single-destination shortest paths:
  - find shortest paths to a destination from every vertex
- Single-pair shortest-path
  - find shortest path from u to v
- All pairs shortest paths

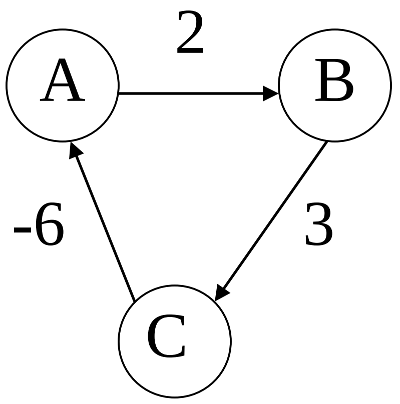
Lemma 24.1

Subpaths of shortest paths are shortest paths.

Given a weighted, directed graph $G = (V, E)$ with weight function $w: E \rightarrow \Re$
Let $p = <v_1, v_2, \ldots, v_k>$ be a shortest path from vertex v_1 to vertex v_k

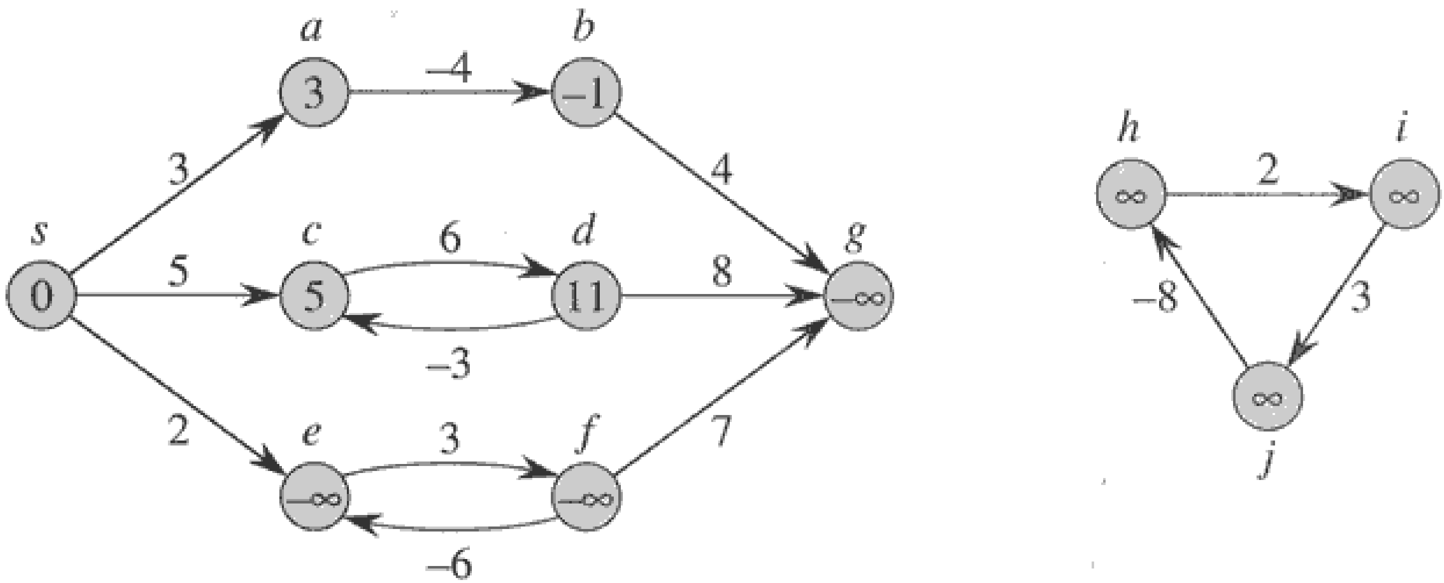
For any i and j such that $1 \leq i \leq j \leq k$, let $p_{ij} = <v_i, v_i+1, \ldots, v_j>$ be a subpath from vertex v_i to vertex v_j.
Then p_{ij} is a shortest path from v_i to v_j.

Negative-Weight Edges



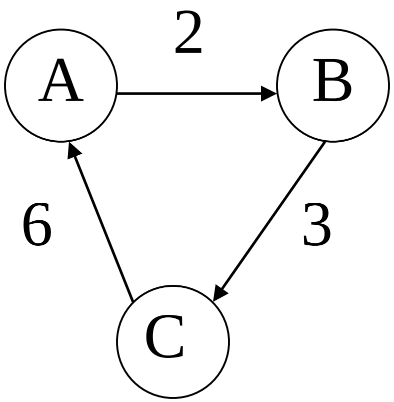
In general, we prohibit edges in our graphs from having negative weights. Look at the cycle above. Going from A to B to C has a cost of 5, but going from C to A has a cost of -6, so the total cost of one cycle from A back to A is -1. But the total cost of two cycles is 0-2, which is less than the cost of one cycle. We can always find a lower-cost path by doing one more cycle!

Negative-Weight Edges



As long as the graph has no negative-weight cycles which are reachable from the source node, we’re OK. But we often just make the assumption that all of the edges have a nonnegative cost.

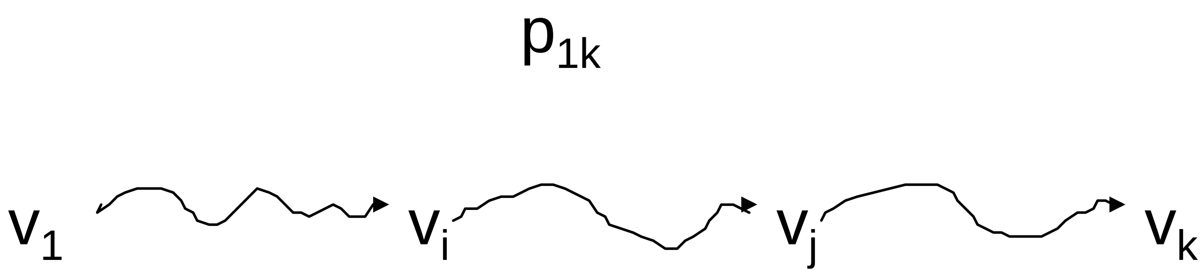
Cycles



Can a shortest path contain a cycle?

No. If we exclude negative cycles, then all cycles will add to the cost of the path, while taking us back to a given node.

Shortest subpath



We can decompose path p_{1k} into several subpaths:

p_{1i} p_{ij} p_{jk}

$w(p_{1k}) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$

Assume p_{1k} is the shortest path from 1 to k.

Then $w(p_{1k})$ is the lowest cost (shortest one) of a path from 1 to k.

Corollary 24.2

Let G = (V,E) w: E → R

Suppose shortest path *p* from a source *s* to vertex *v* can be decomposed into

$$s \xrightarrow{p'} \ldots \rightarrow u \rightarrow v$$

for vertex *u* and path *p'*.

Then weight of the shortest path from *s* to *v* is

$$\delta(s,v) = \delta(s,u) + w(u,v)$$

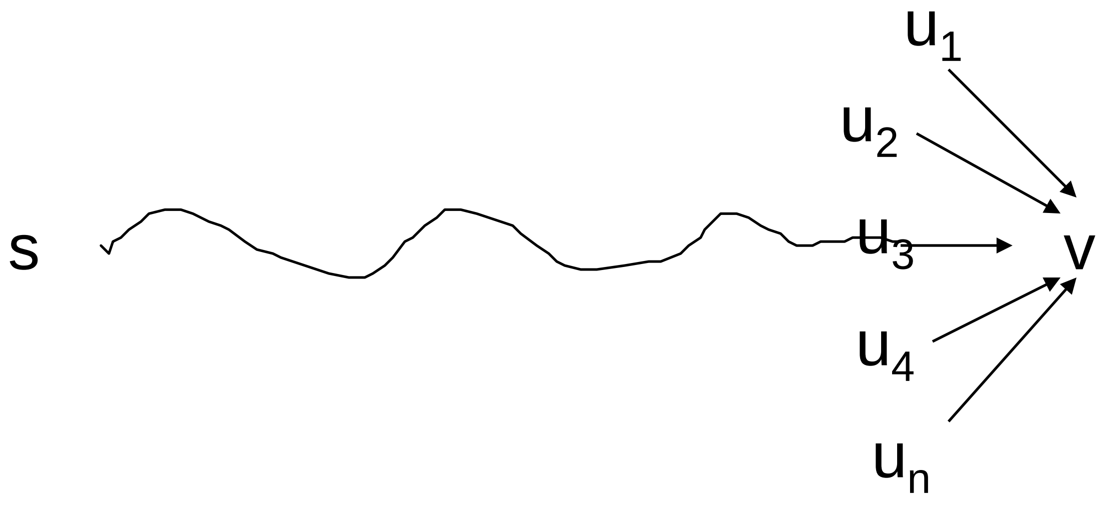
Lemma 24.3

Let $G = (V,E)$ $w: E \rightarrow R$

Source vertex s

For all edges $(u,v) \in E$

$$\delta(s,v) \leq \delta(s,u) + w(u,v)$$



s
u1
u2
u3
v
u4
un

Representing Shortest Paths

- Predecessor subgraph induced by π values: $G_\pi = (V_\pi, E_\pi)$

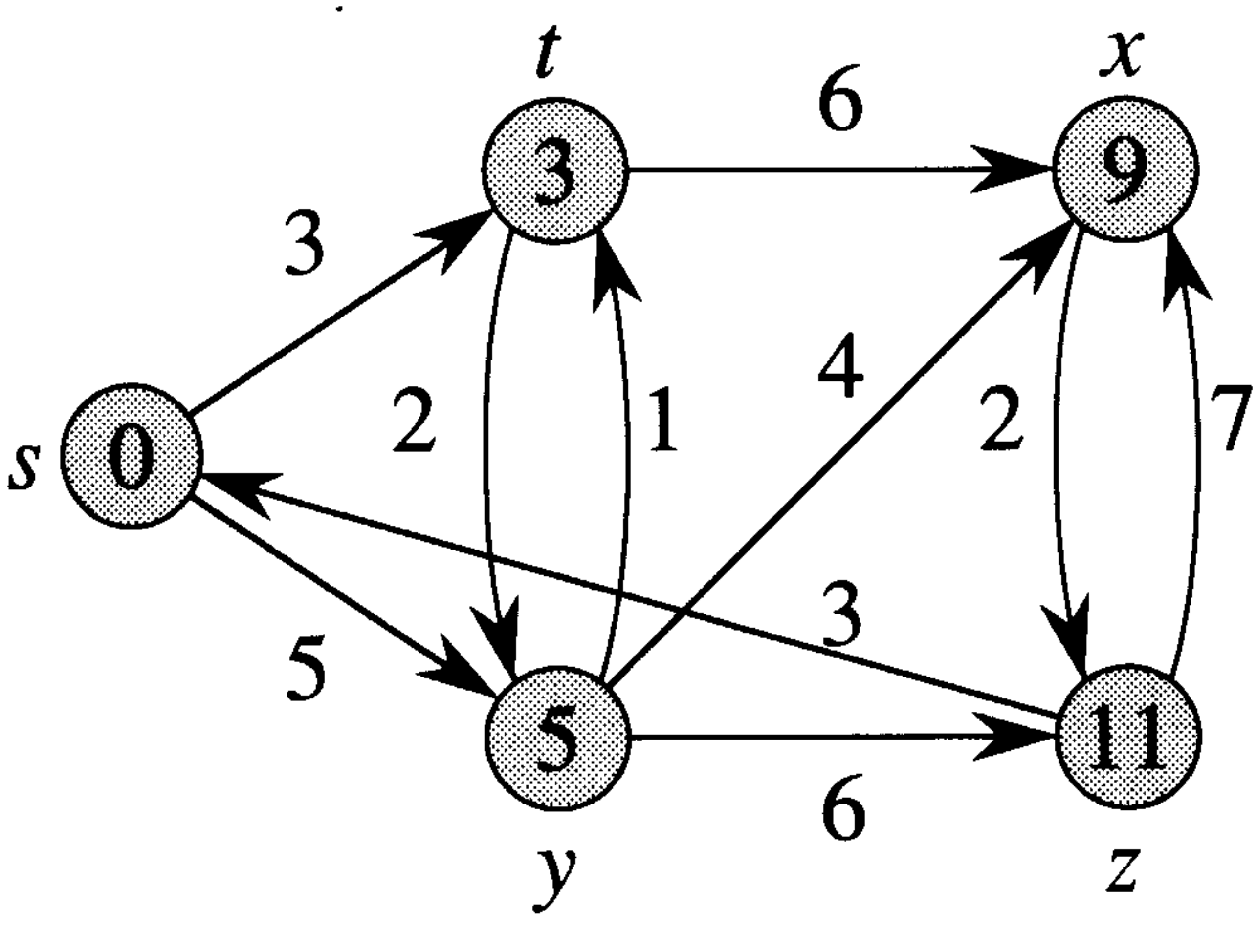
$$V_\pi = \{v \in V: \pi[v] \neq \mathrm{NIL}\} \cup \{s\}$$

$$E_\pi = \{(\pi[v], v) \in E: v \in V_\pi - \{s\}\}$$

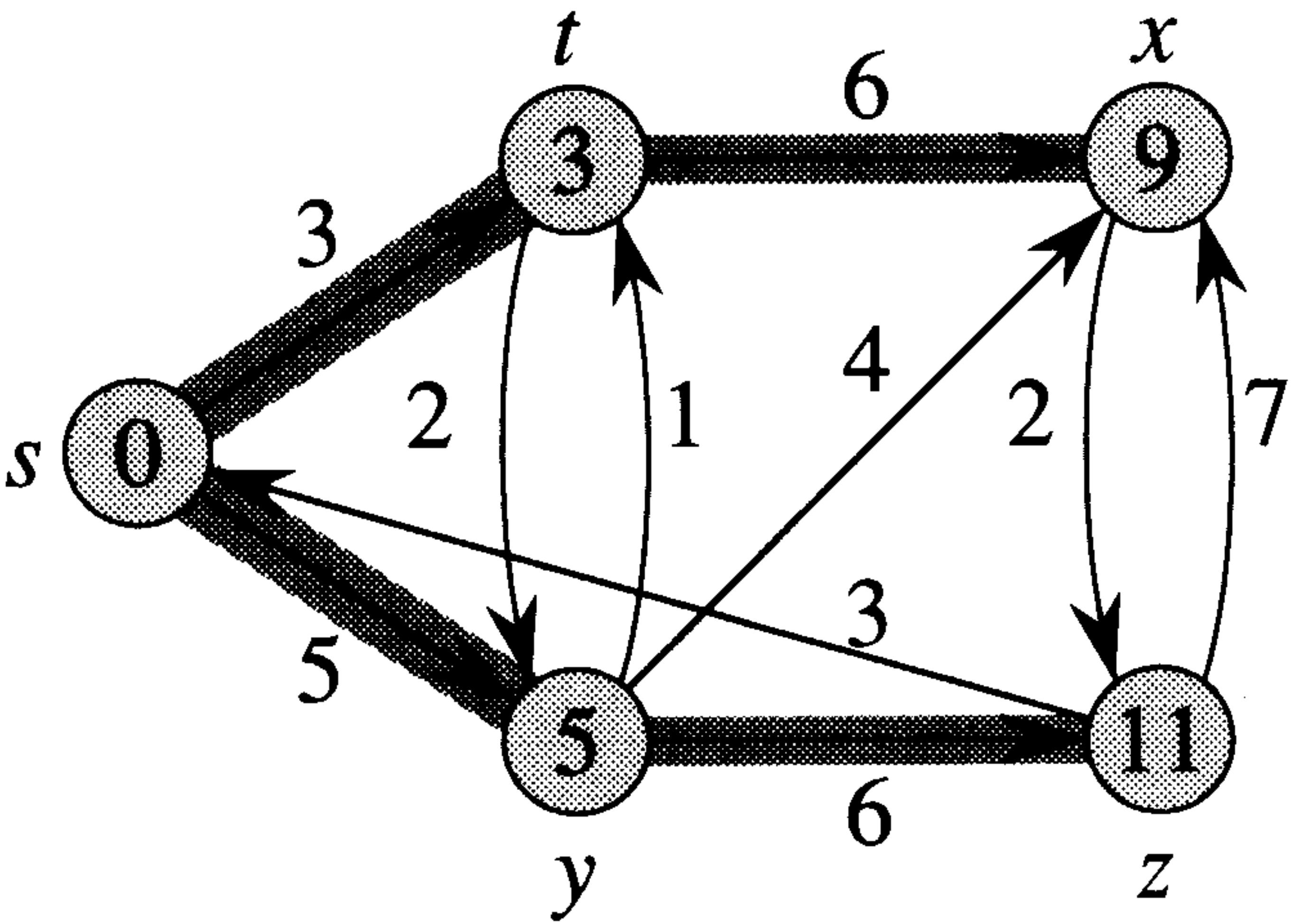
Shortest-Paths Tree

- A shortest-paths tree rooted at *s* is a directed subgraph G´ = (V´, E´), where V´ $\subseteq$ V and E´ $\subseteq$ E, such that
  - V´ is the set of vertices reachable from *s* in G,
  - G´ forms a rooted tree with root *s*, and
  - for all *v* $\in$ V´, the unique simple path from *s* to *v* in G´ is a shortest path from *s* to *v* in G.

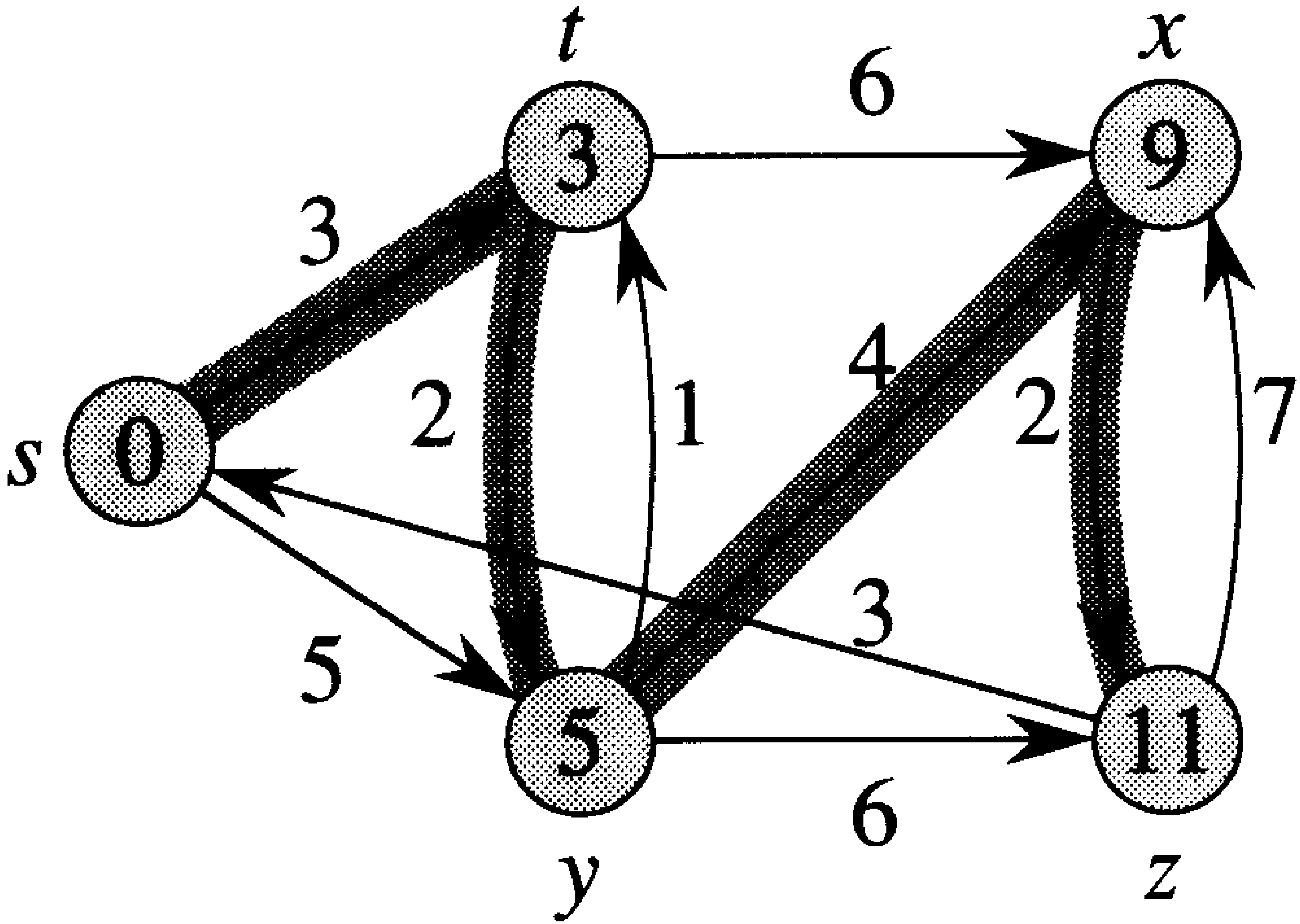
Example of Shortest-Paths Trees



Example of Shortest-Paths Trees



Example of Shortest-Paths Trees



Shortest Path and Relaxation

- Shortest path estimate:

  $d[v]$ is an attribute of each vertex which is an upper bound on the weight of the shortest path from s to v

- Relaxation is the process of incrementally reducing $d[v]$ until it is an exact weight of the shortest path from s to v

Initializing the Shortest-Path Estimates

```
INITIALIZE-SINGLE-SOURCE (G, s)
for each vertex v ∈ V[G] do
    d[v] ← ∞
    π[v] ← NIL
d[s] ← 0
```

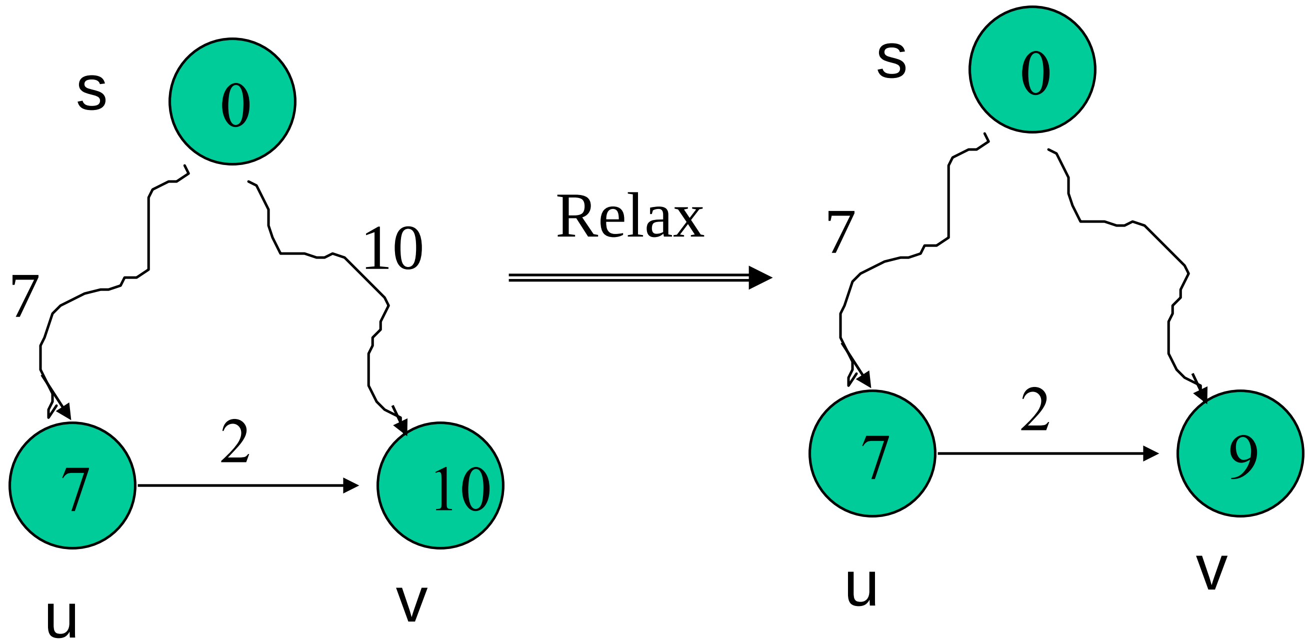
Relaxing an Edge (u,v)

- Question: Can we improve the shortest path to v found so far by going through u?
- If yes, update $d[v]$ and $\pi[v]$

Relaxing an Edge

```
RELAX (u, v, w)
   if d[v] > d[u] + w(u, v)
        then d[v] ← d[u] + w(u, v)
             π[v] ←u
```

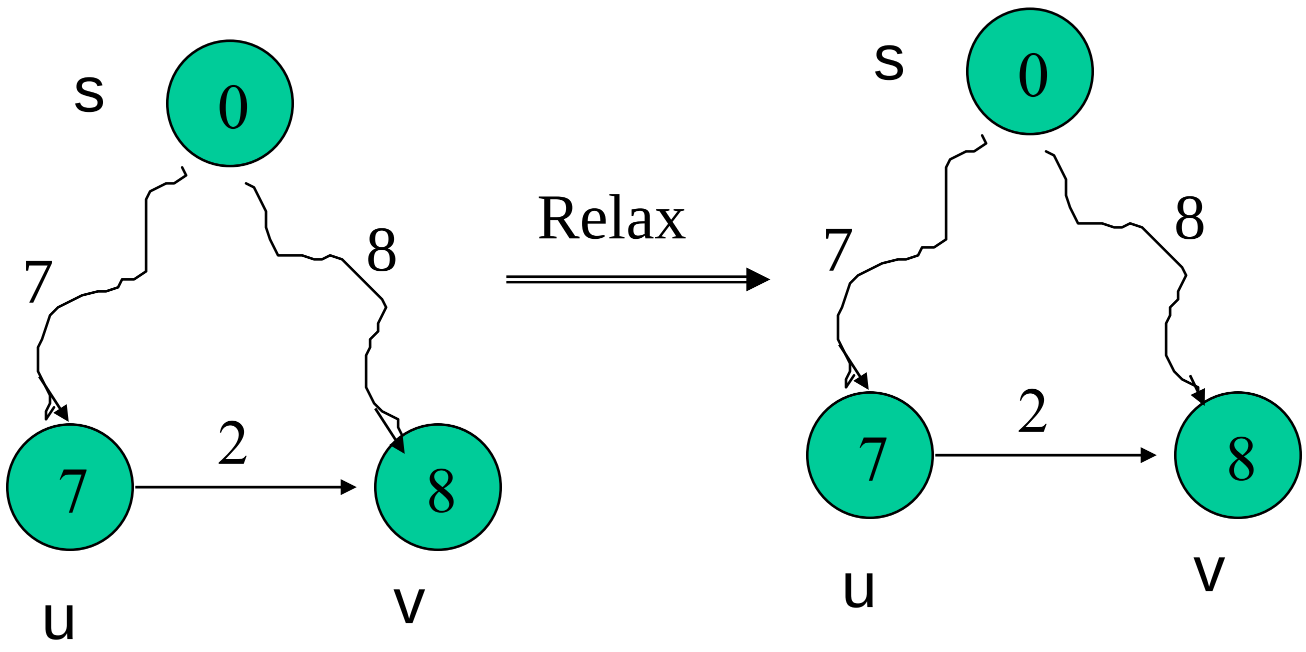
Examples of Relaxation



Before relaxation, $d(v) > d(u) + w(u, v)$

After relaxation, $d(v)$ decreases

Examples (continued)



Before relaxation, $d(v) \leq d(u) + w(u, v)$

After relaxation, $d(v)$ does not change

Dijkstra's Algorithm

- Problem:
  - Solve the single source shortest-path problem on a weighted, directed graph G(V,E) for the cases in which edge weights are non-negative

Dijkstra’s Algorithm

Basic approach:

Maintain a set S of vertices whose final shortest path weights from the source *s* have been determined.

Repeat:

- select vertex u from V-S with the minimum shortest path estimate
- insert u in S
- relax all edges leaving u

Dijkstra's Algorithm

```
DIJKSTRA (G, w, s)
INITIALIZE-SINGLE-SOURCE (G, s)
S ← ∅
Q ← V[G]
while Q ≠ ∅ do
    u ← EXTRACT-MIN(Q)
    S ← S ∪ {u}
    for each vertex v ∈ Adj[u] do
        RELAX(u, v, w)
```

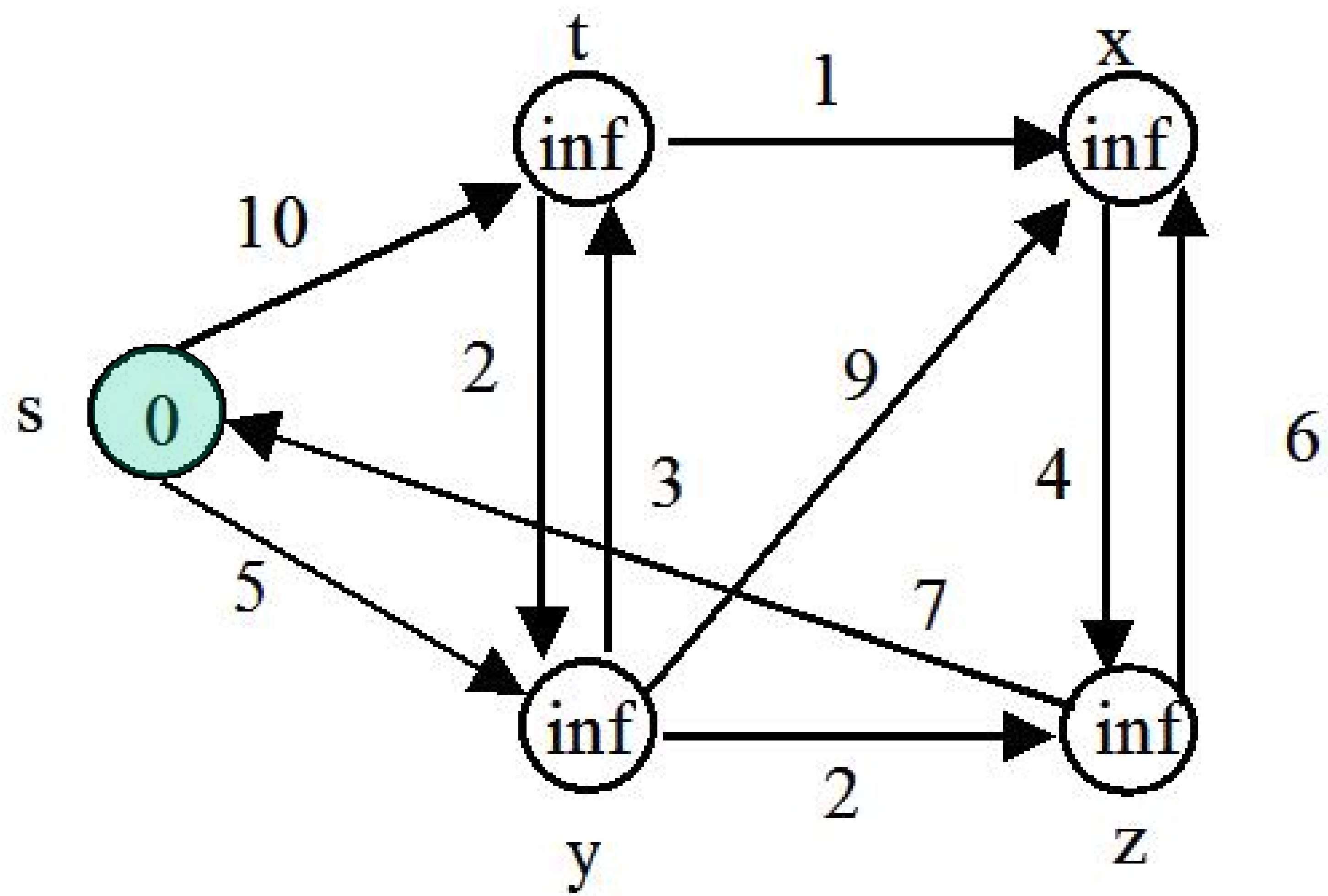
Dijkstra’s Algorithm

Note that in Dijkstra’s algorithm:

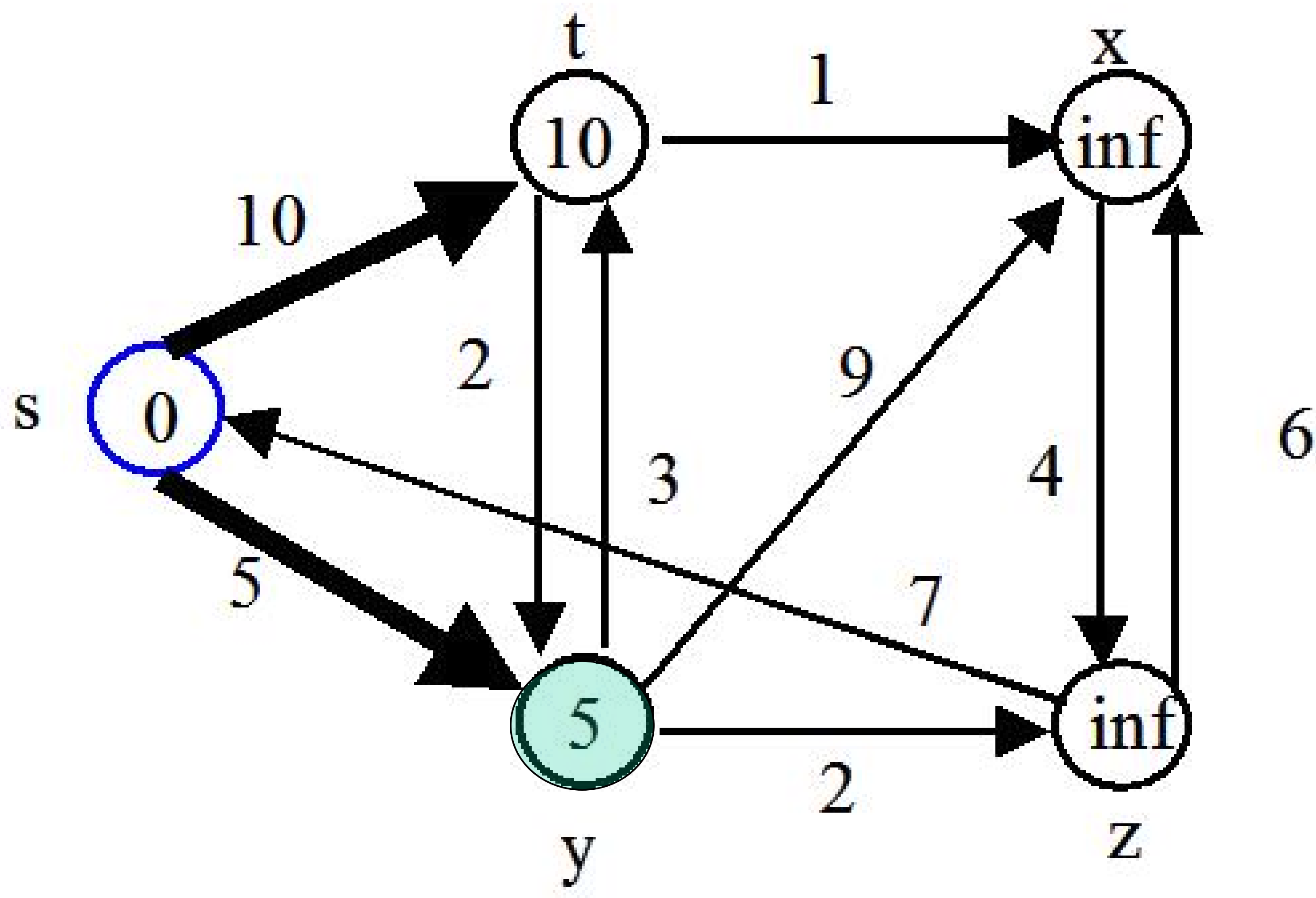
• we select a *single* vertex at each step

• we relax *each* edge leaving that vertex (not just the edge with the lowest weight)

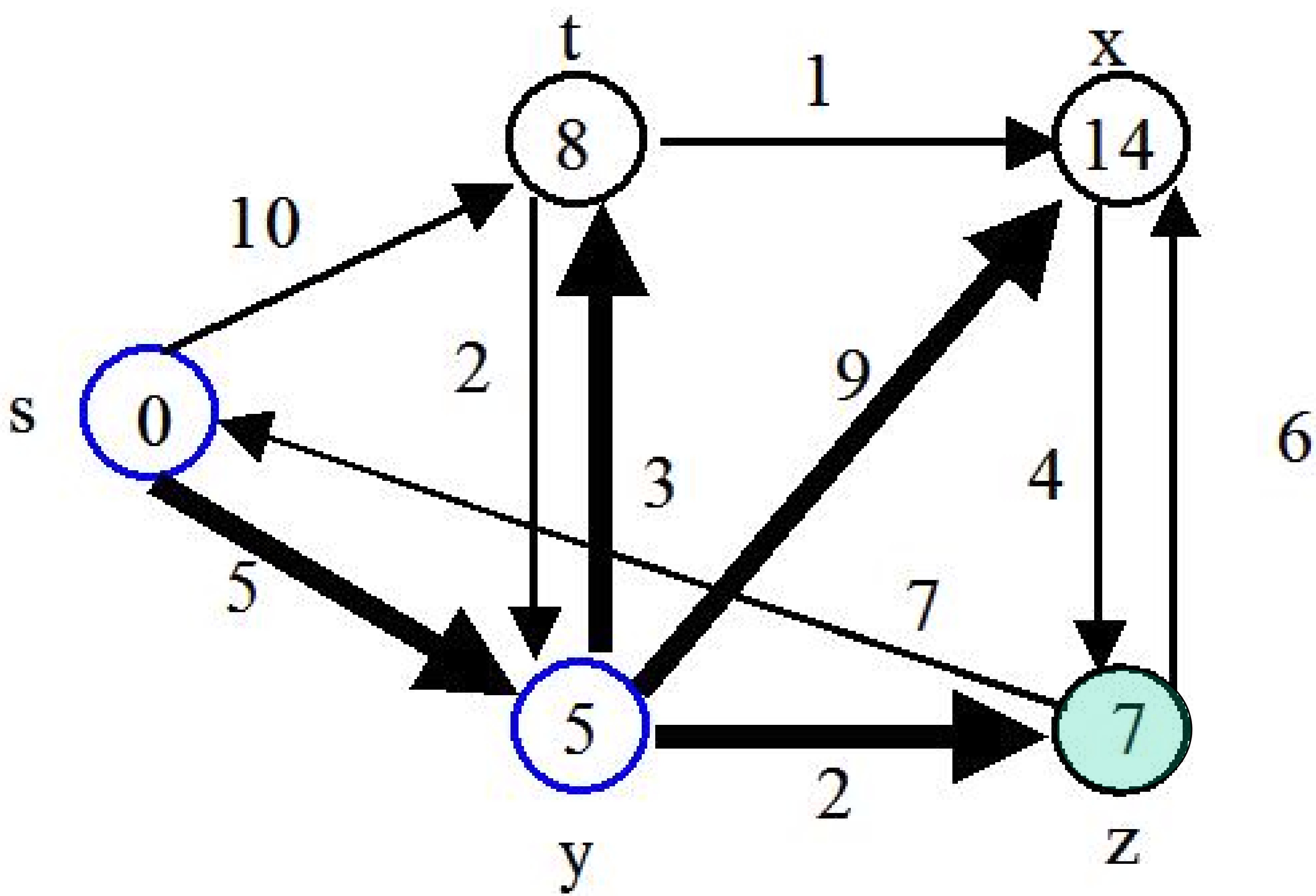
Example of Dijkstra’s Algorithm



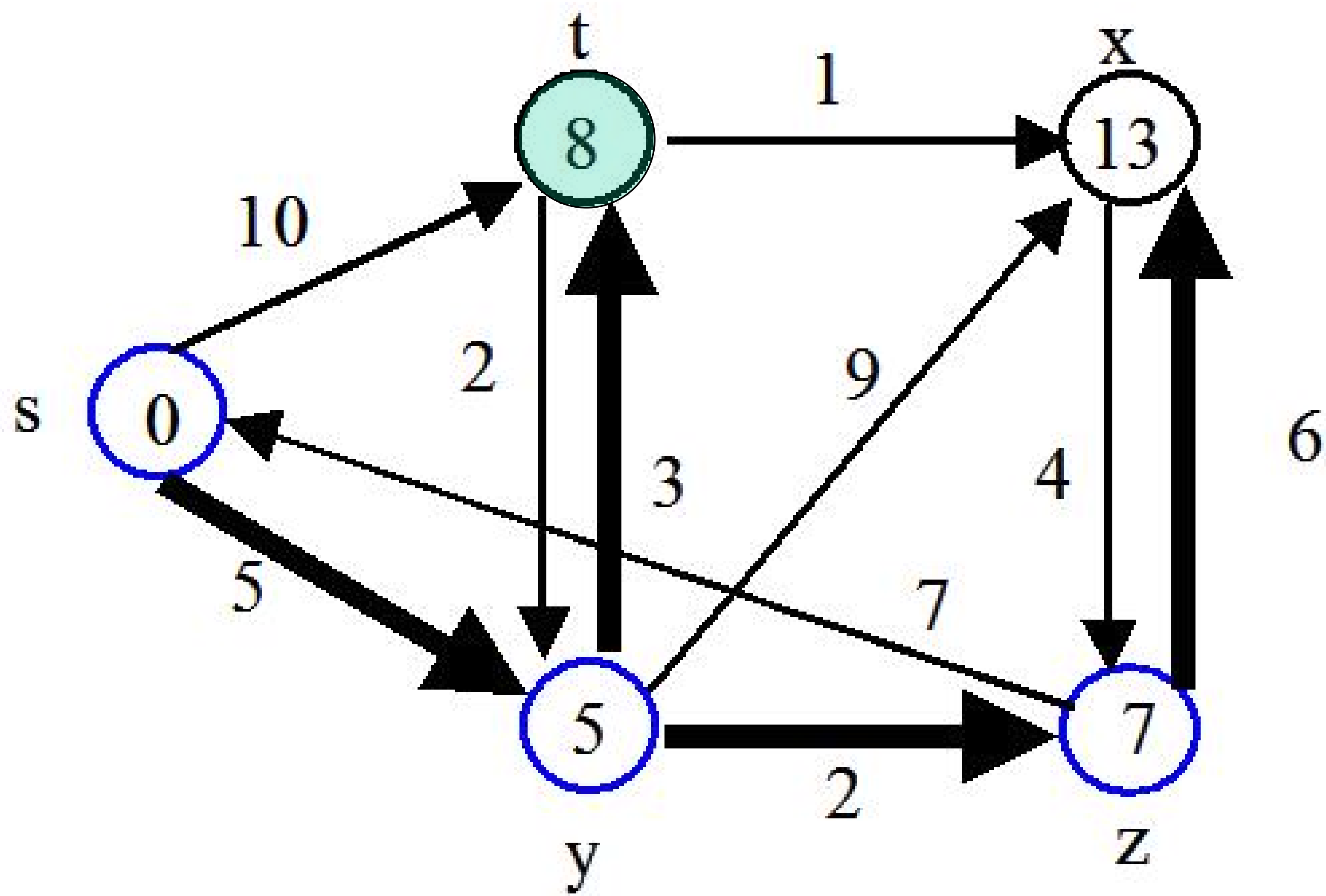
Example (continued)



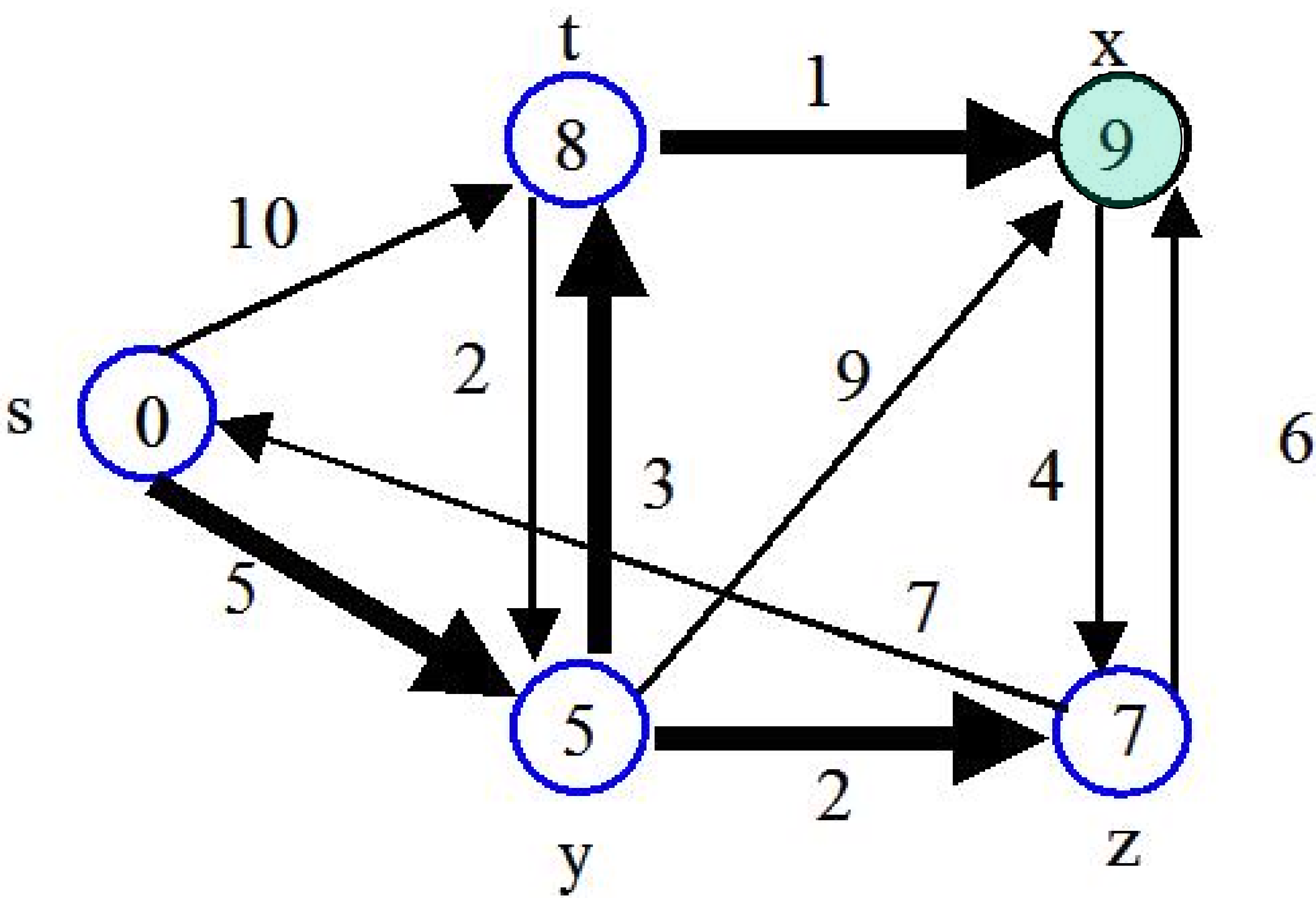
Example (continued)



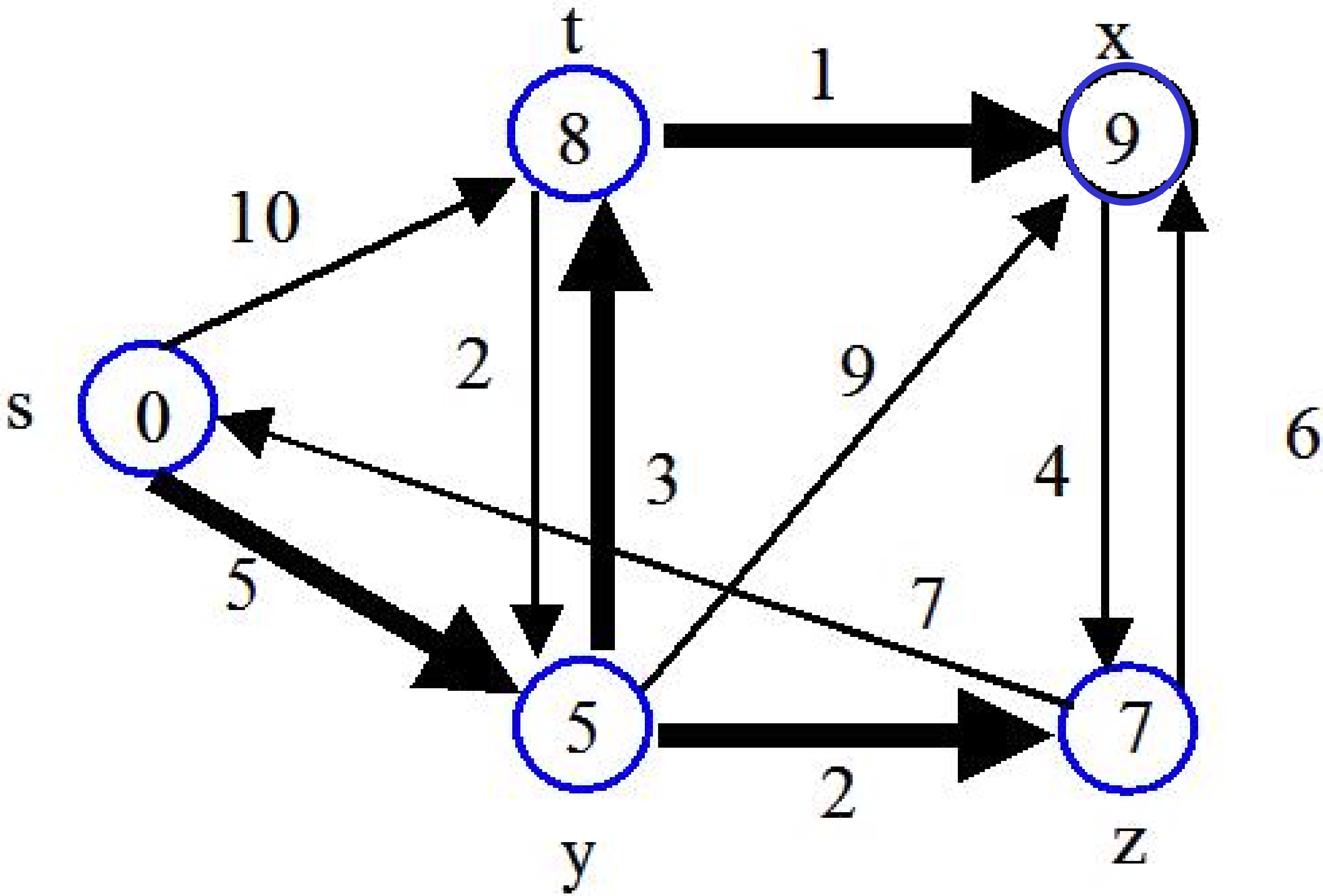
Example (continued)



Example (continued)



Example (continued)



Analysis of Dijkstra's Algorithm

Suppose the priority queue is a binary heap:

BUILD-HEAP	O(V)
Each EXTRACT-MIN	O(lg V)
This is done V times	O(V lg V)
Each edge's relaxation	O(lg V)
Each edge relaxed one time	O(E lg V)
Total time:	O(V lg V + E lg V)

Correctness of Dijkstra's Algorithm

- The correctness of Dijkstra's algorithm can be proved using the following loop invariant:
  - At the start of each iteration of the while loop of lines 4-8, $d[v]=\delta(s,u)$ for each vertex $v \in S$.

Correctness of Dijkstra’s Algorithm

Therefore, the relaxation method, as implemented in Dijkstra’s algorithm, is guaranteed to result in a shortest-paths tree.

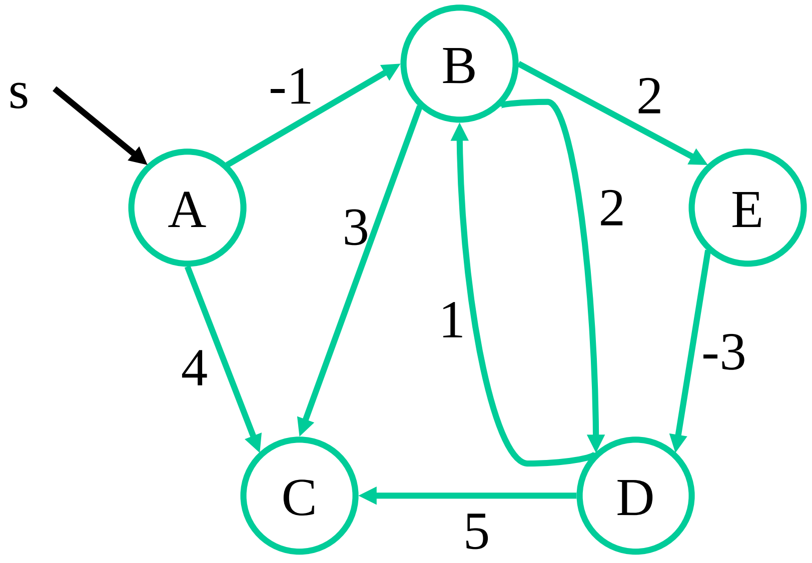
Similarity to Prim's algorithm

Dijkstra's algorithm resembles Prim's in that both algorithms use a min-priority queue to find the "lightest" vertex outside a given set (the set S in Dijkstra's algorithm, and the tree being grown in Prim's algorithm), add this vertex into the set, and adjust the weights of the remaining vertices outside the set accordingly.

Negative-weight edges

- Dijkstra's algorithm assumes all edge weights are non-negative
- What if there are negative edges?

Negative-weight edge example



Negative-weight edges

- If there are negative-edges, we can use Bellman-Ford algorithm.

Bellman-Ford 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));
   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, relaxing each edge

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

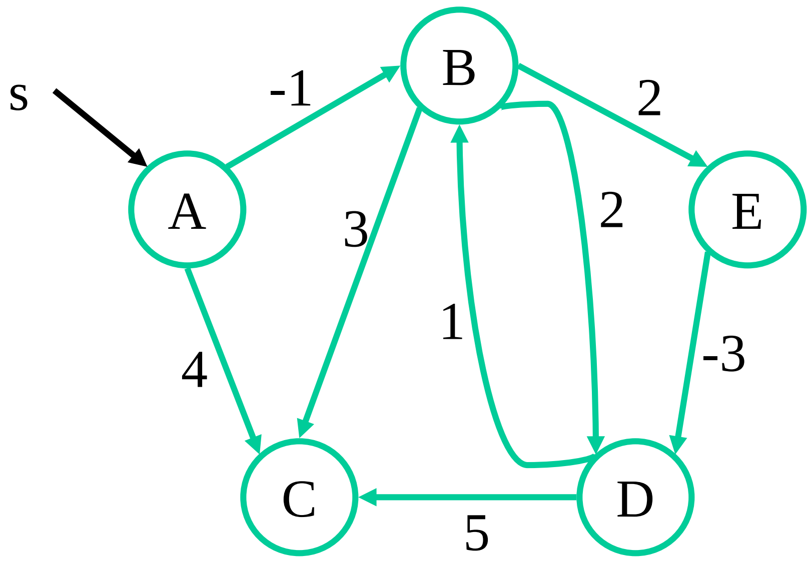
```
Relax(u,v,w): if (d[v] > d[u]+w) then d[v]=d[u]+w
```

Bellman-Ford 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));
   for each edge (u,v) ∈ E
      if (d[v] > d[u] + w(u,v))
           return "no solution";
```

What will be the running time?

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


s
-1
B
2
A
3
2
E
1
-3
4
C
D
5

Bellman-Ford

- Note that order in which edges are processed affects how quickly it converges
- Correctness: show $d[v] = \delta(s,v)$ after $|V|-1$ passes
  - Lemma: $d[v] \geq \delta(s,v)$ always
    - Initially true
    - Let v be first vertex for which $d[v] < \delta(s,v)$
    - Let u be the vertex that caused d[v] to change: $d[v] = d[u] + w(u,v)$
    - Then $d[v] < \delta(s,v)$

      $$\delta(s,v) \leq \delta(s,u) + w(u,v)$$

      $$\delta(s,u) + w(u,v) \leq d[u] + w(u,v)$$
    - So $d[v] < d[u] + w(u,v)$. Contradiction.

Bellman-Ford

- 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