L09.01
Single source shortest path algorithms

50.004 Introduction to Algorithms
Ioannis Panageas (ioannis@sutd.edu.sg)
ISTD, SUTD
CLRS Ch: 24.1 – 24.3
Slides based on Dr Simon LUI

Outline

- **Single source** shortest path problem
- Relaxation
- The Bellman-Ford algorithm
- Dijkstra's algorithm

Definition - weighted path in graph

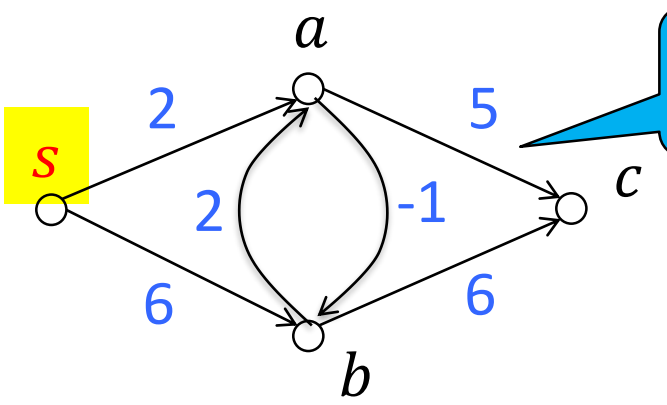
Ingredients:

- A directed graph: $G = (V, E)$
- A weight function: $w: E \to \mathbb{R}$
- A path p $(v_0 \rightsquigarrow^p v_k)$: $p = \langle v_0, v_1, \cdots, v_k \rangle$
- The weight of path p: $w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$

Shortest-paths problems

The weight of the shortest-path $u \rightsquigarrow v$

$$\delta(u,v) = \begin{cases} \min\{w(p) \mid u \overset{p}{\rightsquigarrow} v\} & \exists p: u \overset{p}{\rightsquigarrow} v \\ \infty & \text{otherwise} \end{cases}$$



s is the source state

via node a

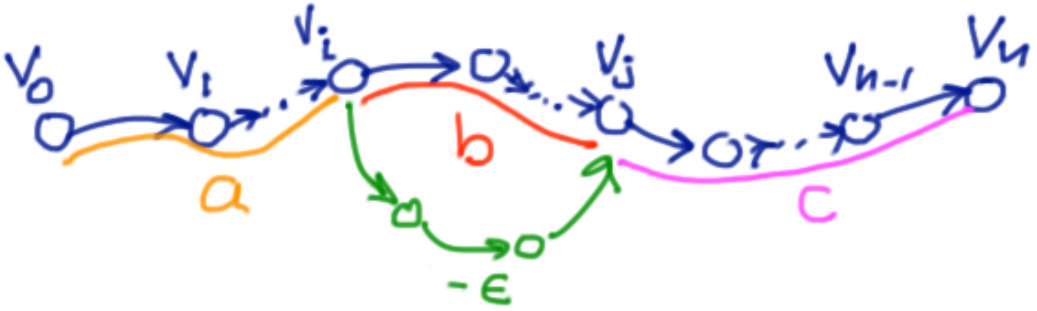
$$\delta(s,b) = \min\{w(\langle s,a,b\rangle), w(\langle s,b\rangle)\} = \min\{1,6\} = 1$$

$$\delta(s,c) = \min\{w(\langle s,a,c\rangle), w(\langle s,a,b,c\rangle), w(\langle s,b,c\rangle)\} = \min\{7,7,12\} = 7$$

via node a or b

An important property

Subpaths of shortest paths are also shortest paths



Proof:

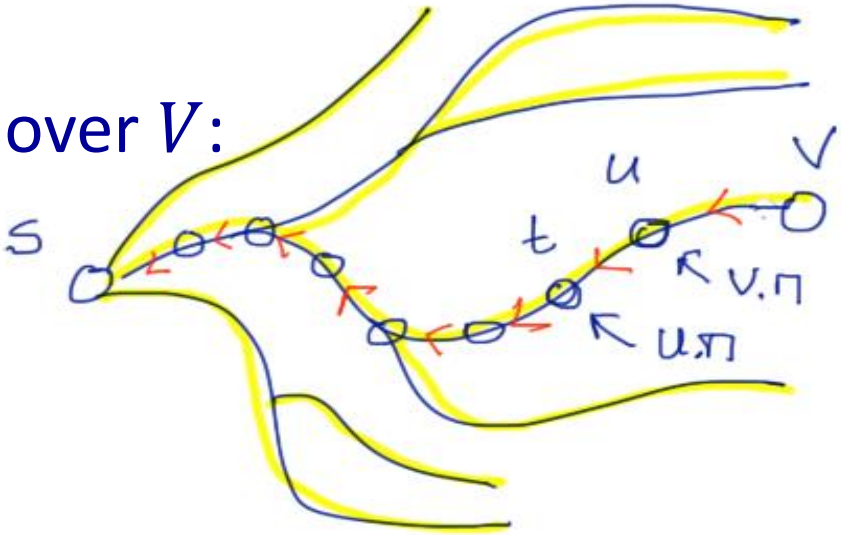
if subpath v_i to v_j is not optimal, but v_0 to v_n is,
then a shorter path from v_i to v_j can also improve the path from v_0 to v_n
….CONTRADICTION!

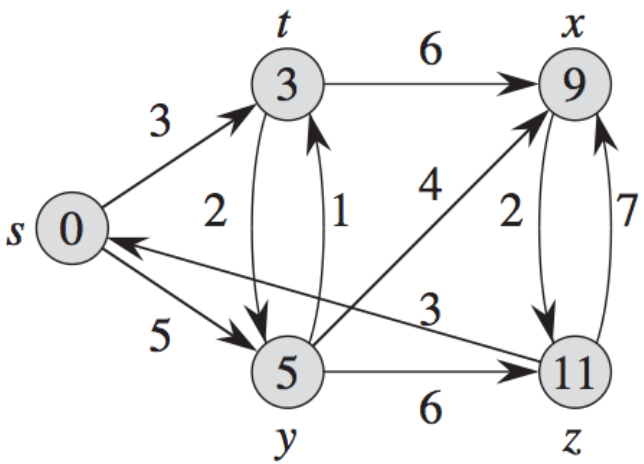
On a shortest path $\langle v_0, \cdots, v_n \rangle$ for $0 \leq j \leq n$:

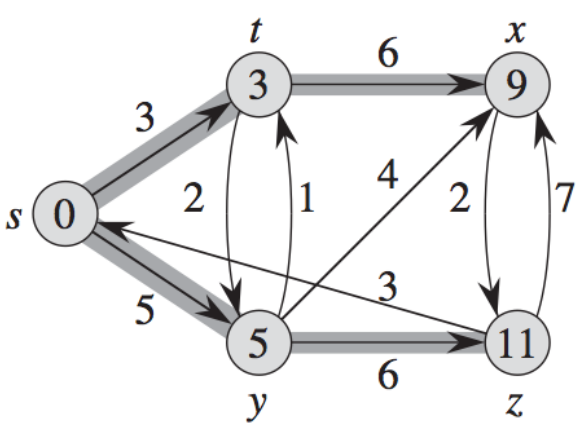
$$\delta(v_0, v_j) + \delta(v_j, v_n) = \delta(v_0, v_n)$$

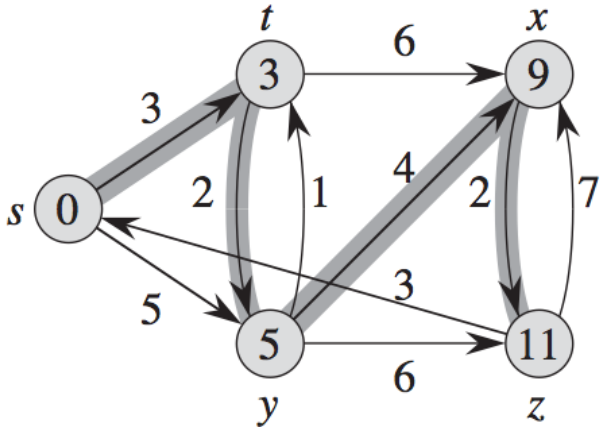
Consequence

$\delta(s, v)$ defines a tree structure over V:
the shortest-path tree
(not necessarily unique)









two possible shortest path trees
but the value function is unique

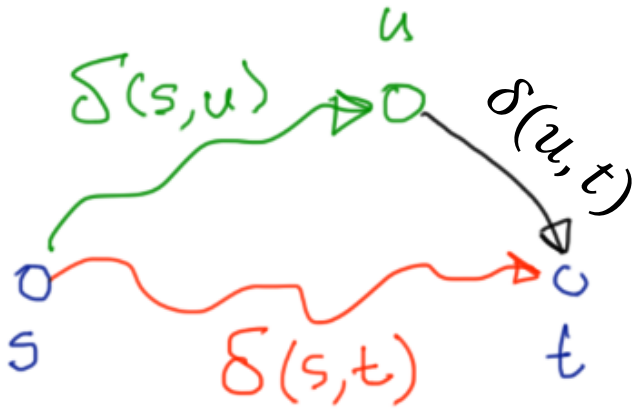
Shortest-paths problems

The single source shortest-path problem:

Determine, given a source node s, for each node v:

1. the shortest path distance $\delta(v) = \delta(s, v)$; and
2. the predecessor node $v.\pi$ along the selected shortest path.

The triangle inequality

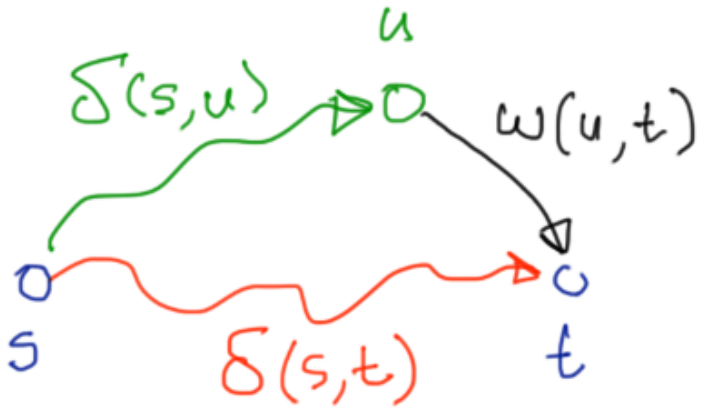


If u is NOT on a shortest path: $\delta(s,t) < \delta(s,u) + \delta(u,t)$

If u is on a shortest path: $\delta(s,t) = \delta(s,u) + \delta(u,t)$

In general: $\delta(s,t) \leq \delta(s,u) + \delta(u,t)$

The triangle inequality



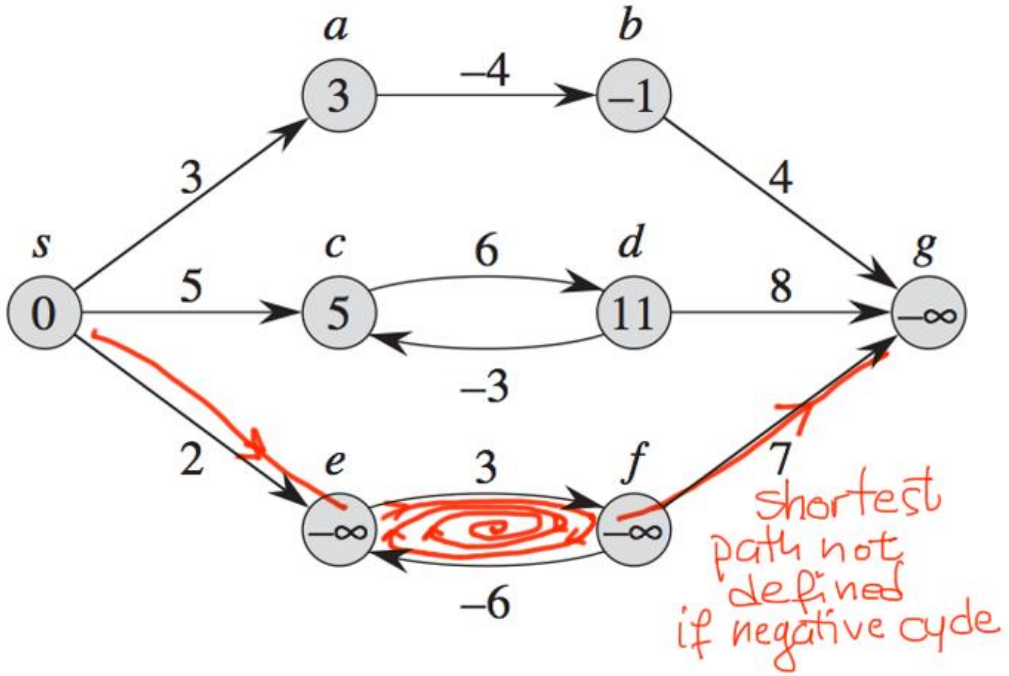
The particular case that $d(u,t) = w(u,t)$ (single edge path)

If u is NOT on a shortest path: $\delta(s,t) < \delta(s,u) + w(u,t)$

If u is on a shortest path: $\delta(s,t) = \delta(s,u) + w(u,t)$

In general: $\delta(s,t) \leq \delta(s,u) + w(u,t)$

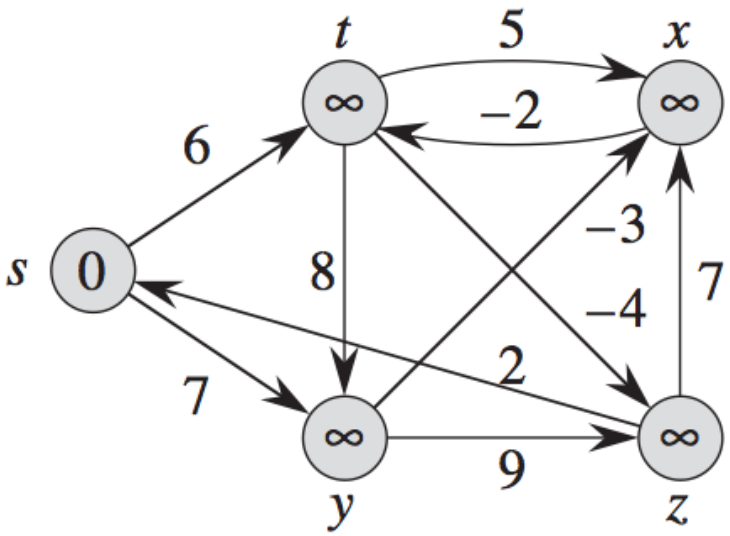
Negative weight edges, negative and positive cycle



1. Negative-weight cycle: if a negative cycle exists, the shortest path cannot be defined. Why?

2. Positive-weight cycle: never occurs in any shortest path. Why?

Questions



Are there negative cycles?

$\delta(s, z) = ?$

Shortest path to $z = ?$

There are NO negative cycles

$\delta(s, z) = -2$

Shortest path to $z = \langle s, y, x, t, z \rangle$

Wrong answers:

$\delta(s, z) = 2$ (WRONG)

Shortest path to $z = \langle s, t, z \rangle$

Idea for a shortest path algorithm

1. Estimate the distance $\delta(v) = \delta(s, v)$ and predecessor $v.\pi$ for each node v

2. Improve the estimates iteratively using distances of surrounding nodes (relaxations)

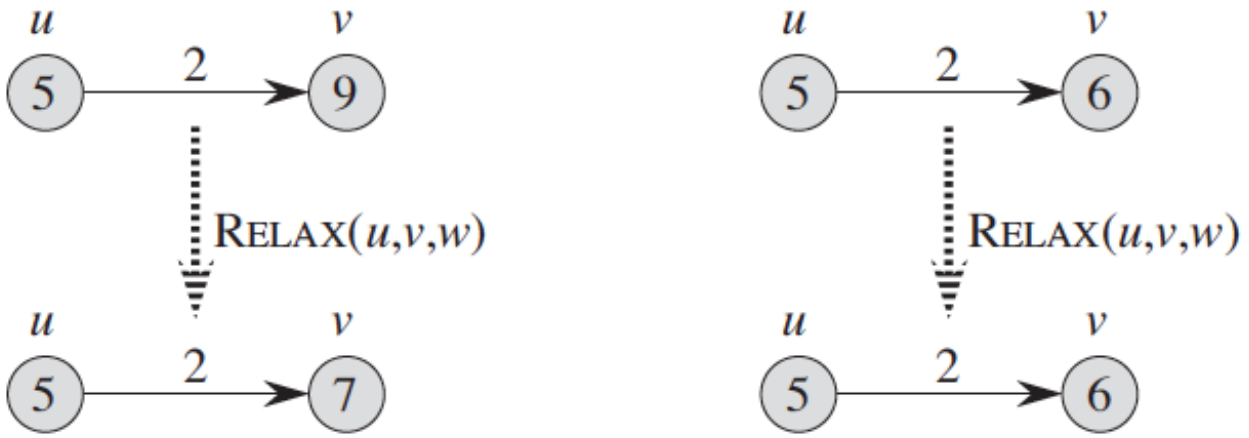
3. Stop when no futher improvement is possible

Initial estimates:

- $\delta(s) = 0$;
- for all $v \neq s$: $\delta(v) = \infty$
- for all v: $v.\pi = \text{nil}$

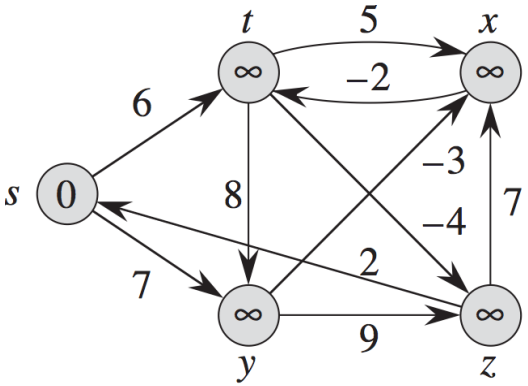
Relaxations

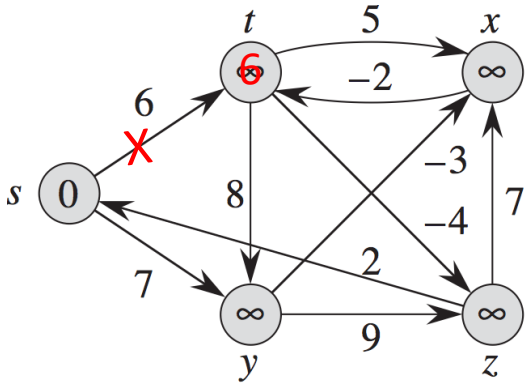
Relaxing an edge (u,v): test if we can improve the shortest path to v (with the value that we found so far), by going through u

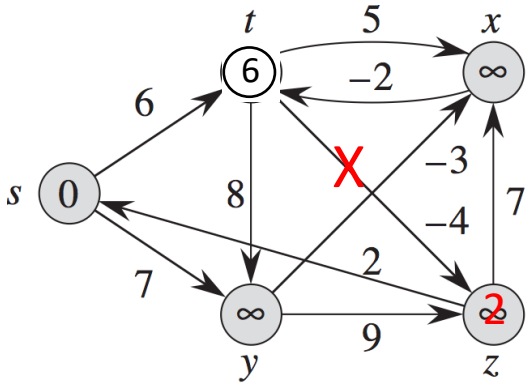


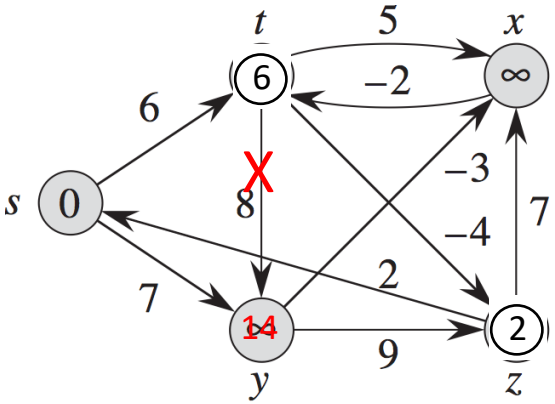
if $\delta(u) + w(u,v) < \delta(v)$ **then**
$\delta(v) := \delta(u) + w(u,v); v.\pi := u$
fi

Iterations using relaxations









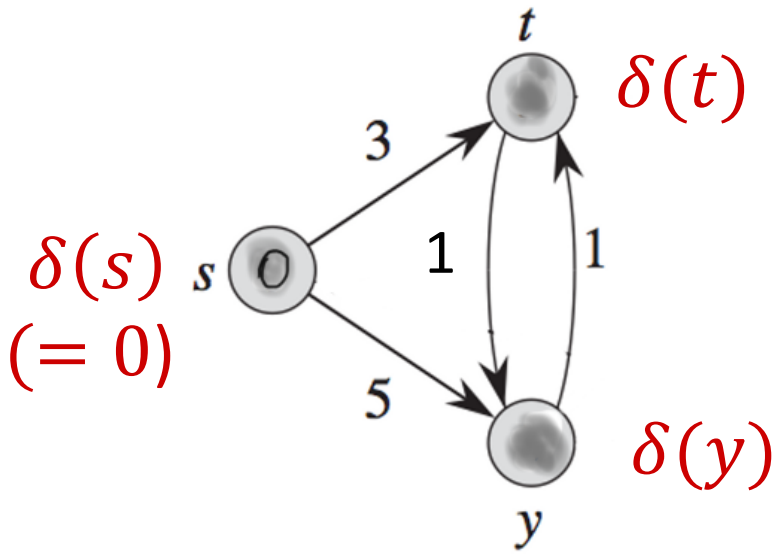
Edges that successfully relaxed are red

Optimal distances as fixpoints

We can characterize the optimal distance $\delta(v)$ by a set of recursive equations that follow the structure of the graph:

these are known as the Bellman equations

Example: $\delta(t)$ and $\delta(y)$



$$\delta(s) = 0$$

$$\delta(t) = \min\{\delta(s) + 3, \delta(y) + 1)\}$$

$$\delta(y) = \min\{\delta(s) + 5, \delta(t) + 1)\}$$

Richard E. Bellmann (1920-1984)

American applied mathematician, who introduced dynamic programming in 1953.

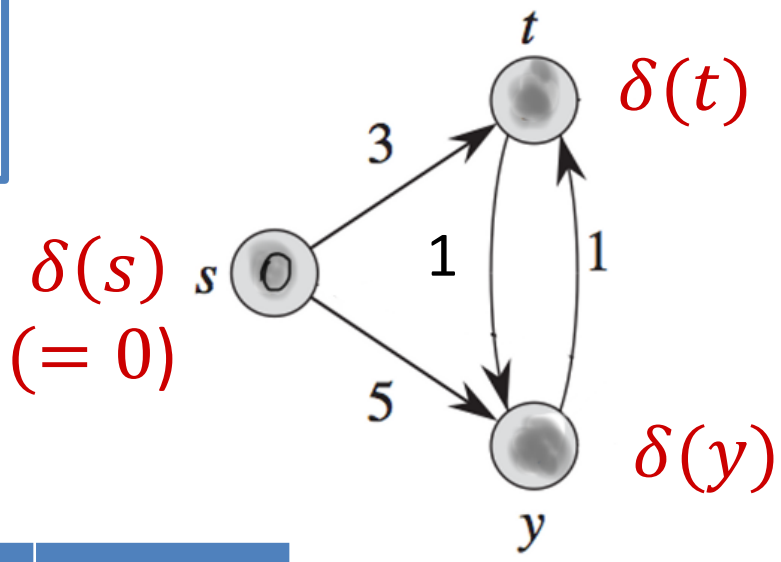
By Source, Fair use, https://en.wikipedia.org/w/index.php?curid=43193672

Finding fixpoints by iterative relaxation

$$\delta(s) = 0$$

$$\delta(t) = \min\{\delta(s) + 3, \delta(y) + 1)\}$$

$$\delta(y) = \min\{\delta(s) + 5, \delta(t) + 1)\}$$



Iterations (time = 0, 1, 2…)

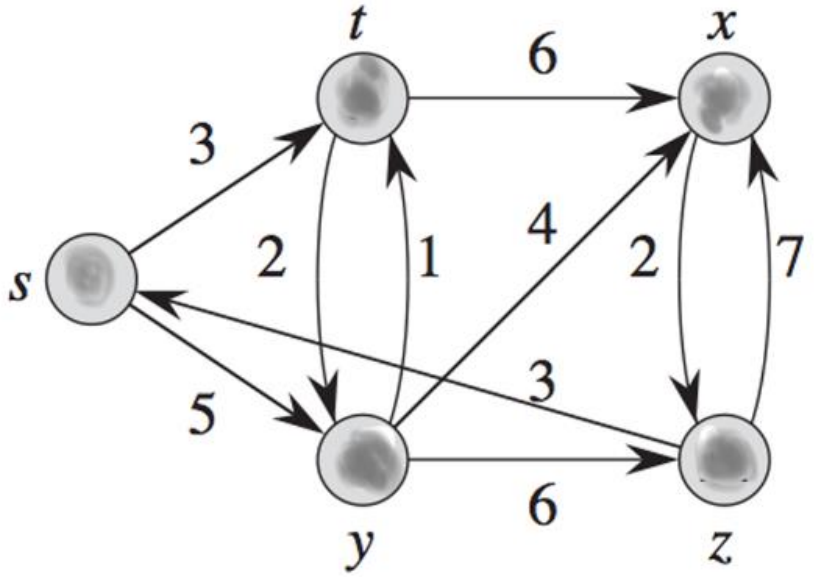
Each node picks a revised predecessor

	time=0	1	2	3
δ(s)	0	0	0	0
δ(t)	∞	3	3	3
δ(y)	∞	5	4	4

no improvement signals termination

Question

Write the Bellman equations for the shortest paths from s for this DG.



$$
\begin{aligned}
&d(s) = 0,\\
&d(t) = \min\{d(s)+3, d(y)+1\},\\
&d(y) = \min\{d(s)+5, d(t)+2\},\\
&d(x) = \min\{d(t)+6, d(y)+4, d(z)+7\},\\
&d(z) = \min\{d(x)+2, d(y)+6\}
\end{aligned}
$$

Solving the Bellman equations

For each node repeat until equilibrium is reached (distance estimates no longer improve):

1. inspect updated shortest distance estimates from preceding neighbours;
2. pick predecessor that can minimize its shortest distance though relaxation;
3. update estimation.

Cost: number of relaxations we need to perform.

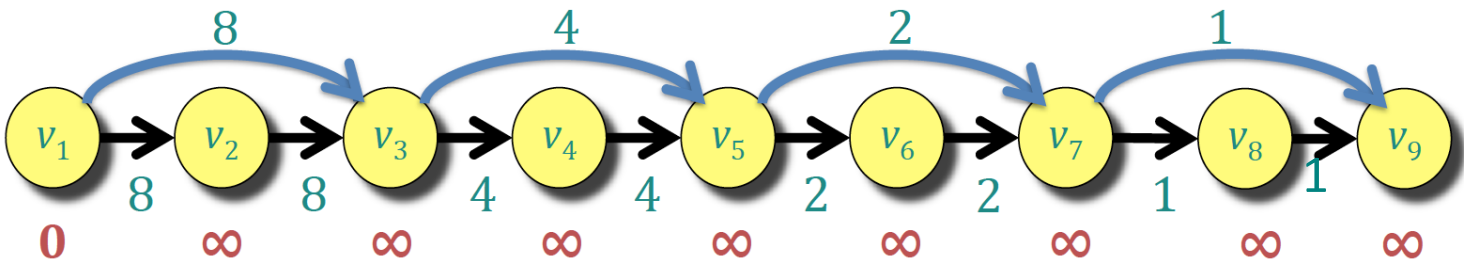
How many relaxations are needed?

1. Minimally: $|V| - 1$, if we relax along the shortest path tree in the right order (cf. the useful observation before).
2. Maximally: exponential in the number of edges $|E|$: see following example.

Smart solutions for selecting relaxations:

- Bellman-Ford algorithm
- Dijkstra's algorithm

Example with exponential #relaxations



8 16 20 24 26 28 29 30

29

26 27 28

27

20 22 24 25 26

25

22 23 24

23

$$T(n) = 3 + 2.T(n-2) \sim \Theta(2^{n/2})$$

8 12 16 18 20 21 22

21

18 19 20

19

12 14 16 17 18

17

14 15 16

15

Key idea Bellman-Ford algorithm

After initialization relax all edges $|V| - 1$ times, i.e.

```
for v in V:
      v.d = ∞          } initialization
      v.π = nil
s.d = 0
do n-1 times:
     for each edge (u,v) in E:   } main loop
          relax (u,v)
```

Then the edges of all shortest paths (if they exist) are relaxed at least once in the order of the shortest path tree. If we can show that the relaxations of one path do not interfere with those of other paths, we are done.

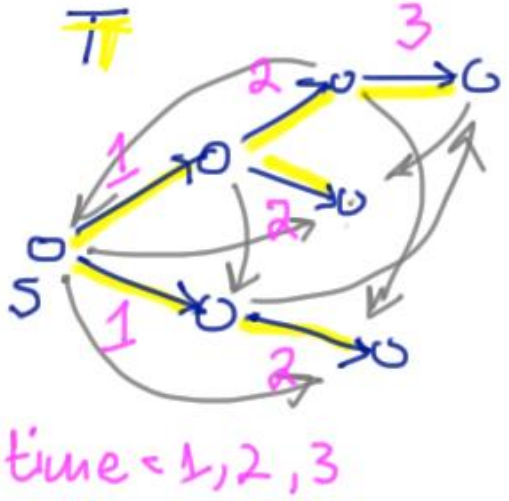
Negative-weight cycles

If there are negative-weight cycles, then there are vertices v whose value of $d[v]$ can still be reduced after $|V| - 1$ iterations of the main loop.

Which vertices and why?

This can be used to detect negative cycles.

Bellman-Ford



for v in V:
    $v.d = \infty$
    $v.\pi$ = nil
$s.d = 0$

initialization
$\Theta(|V|)$

do $n-1$ times:
    for each edge (u,v) in E:
        relax (u,v)

main loop
$\Theta(|V||E|)$

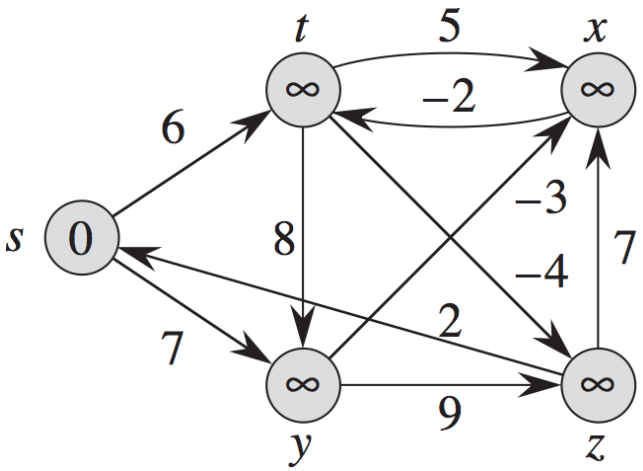
Complexity: $\Theta(|V||E|)$
$\Theta(|V|3)$: dense graphs
$\Theta(|V|2)$: sparse graphs

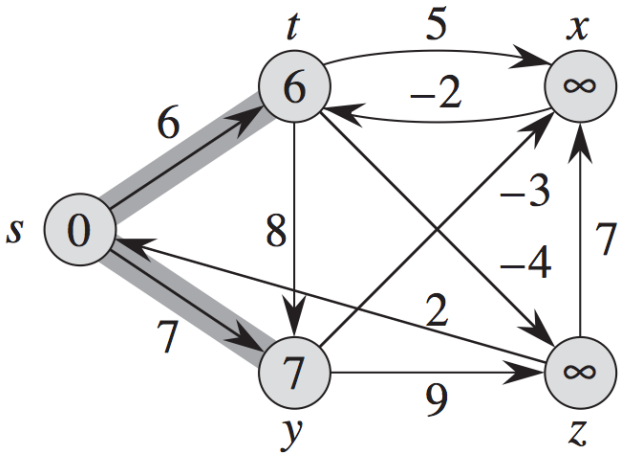
for each edge (u,v) in E:
    if $v.d > u.d + w(u,v)$:
        report a negative cycle & return
report no negative cycles

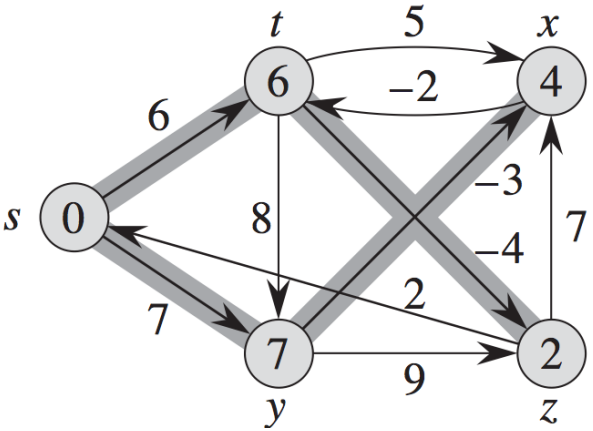
negative cycle detection
$\Theta(|E|)$

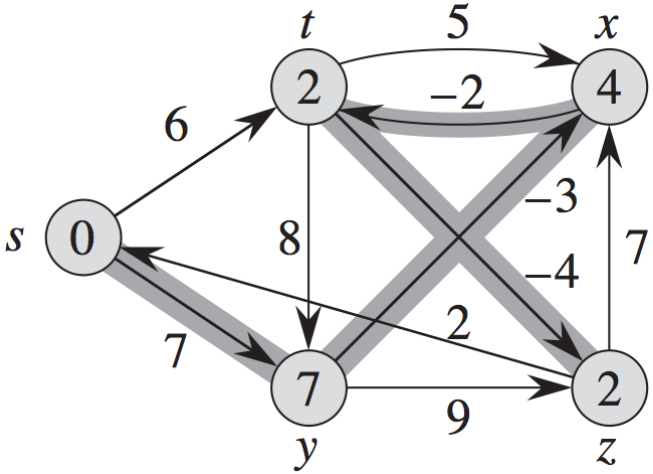
Property: if no negative cycles, then at termination for all v: $v.d = \delta(s,v)$

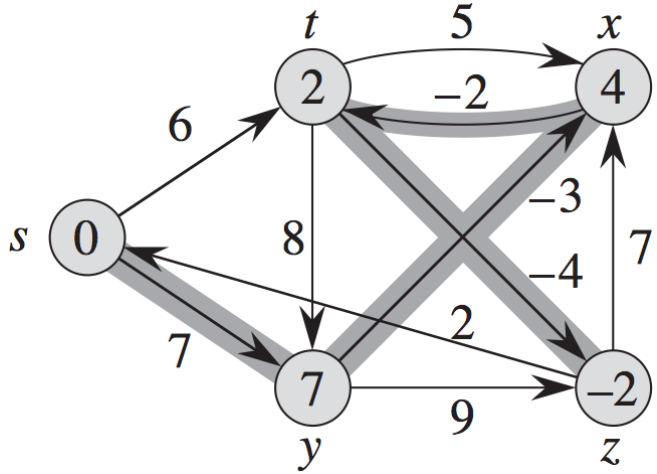
Bellman-Ford example



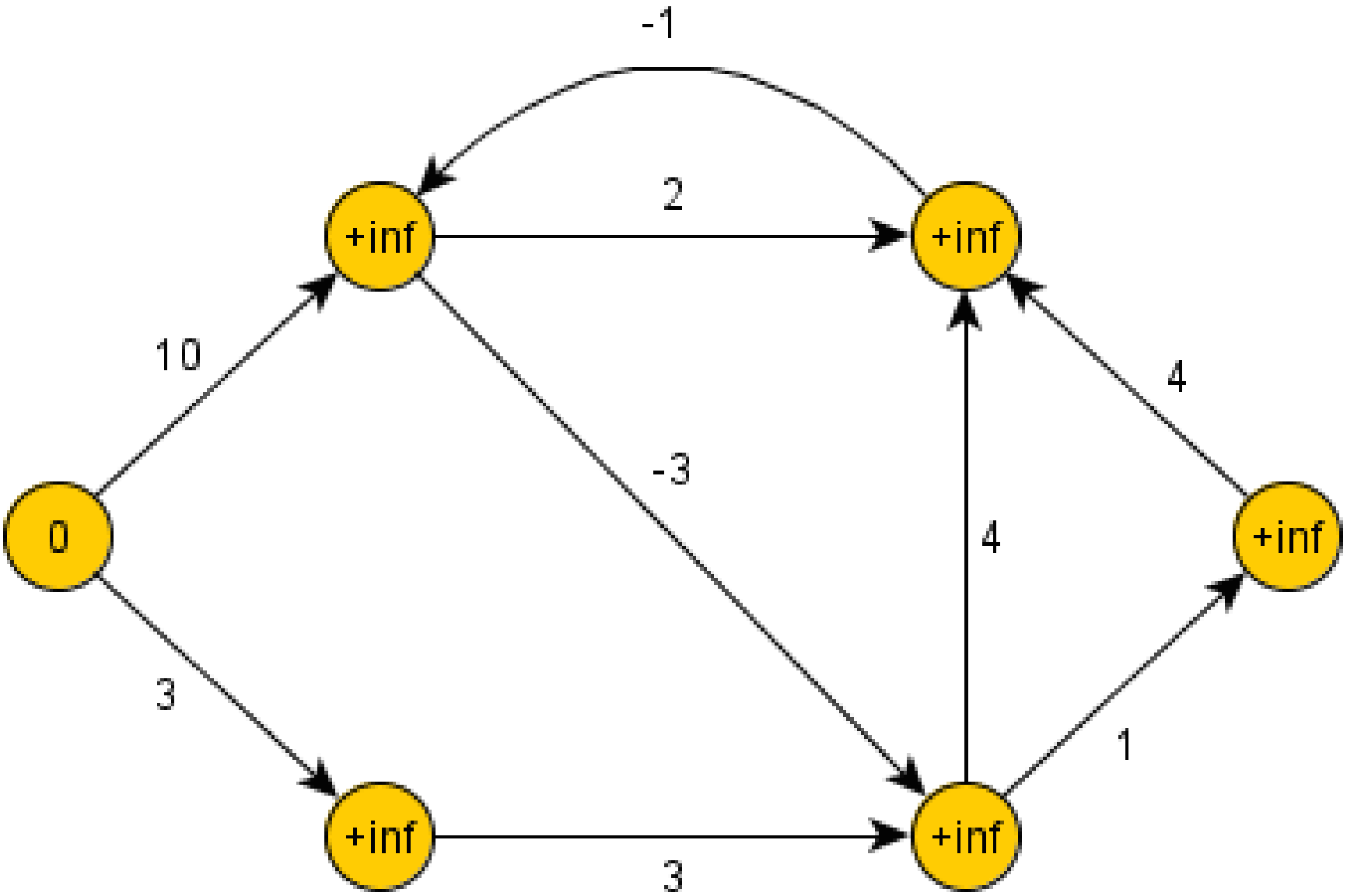






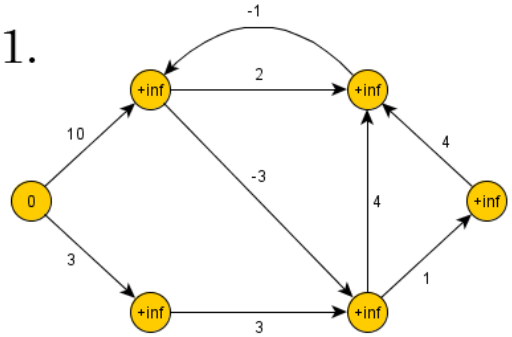


Exercise

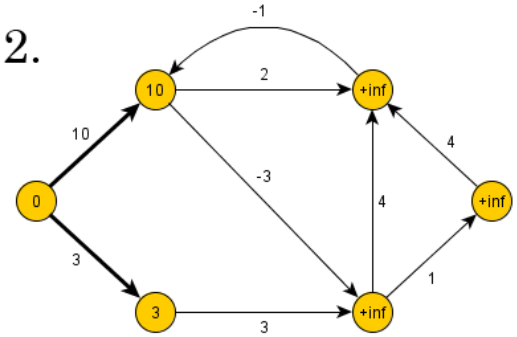


Answer

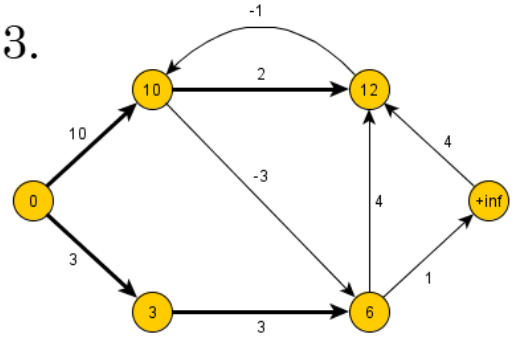
1.



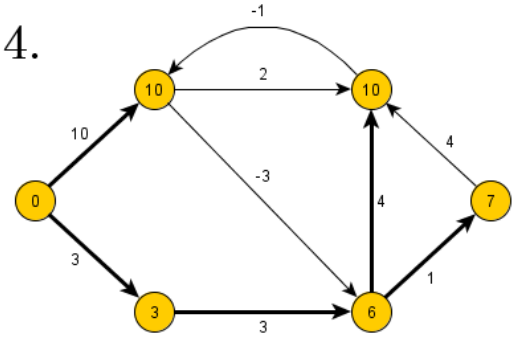
2.



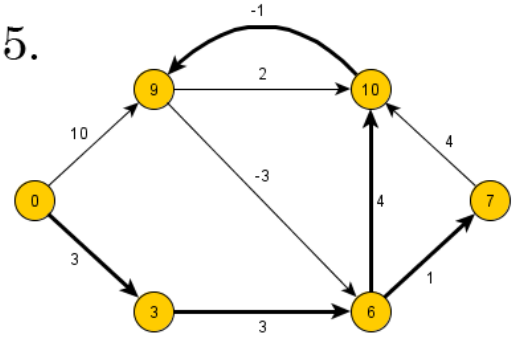
3.



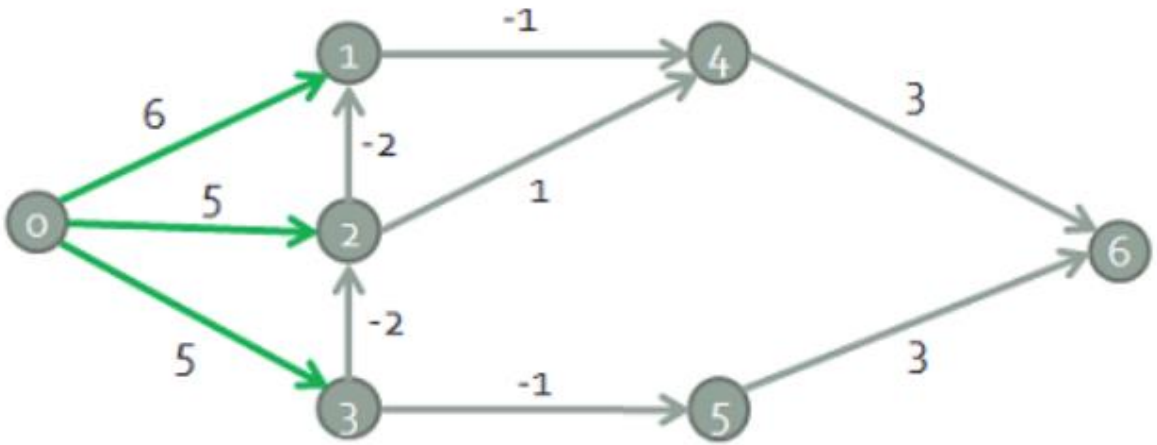
4.



5.

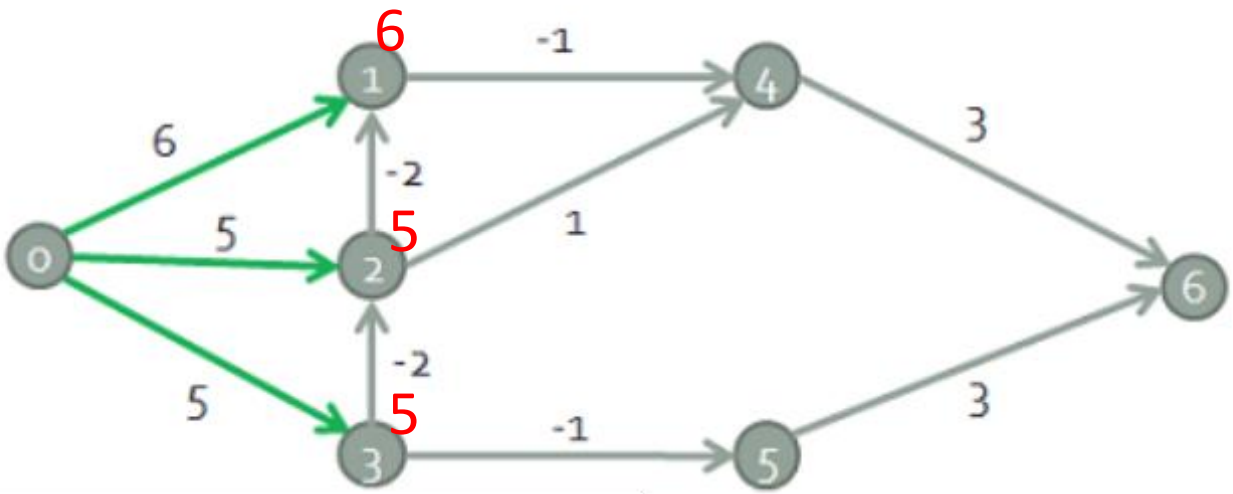


Example 2



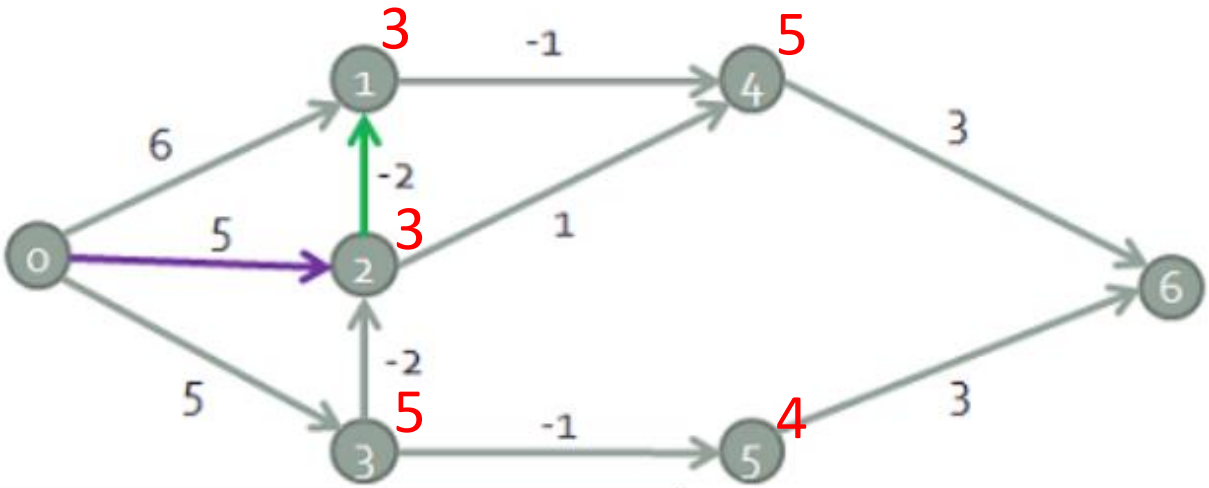
Find the shortest path from node 0

Example 2 Answer



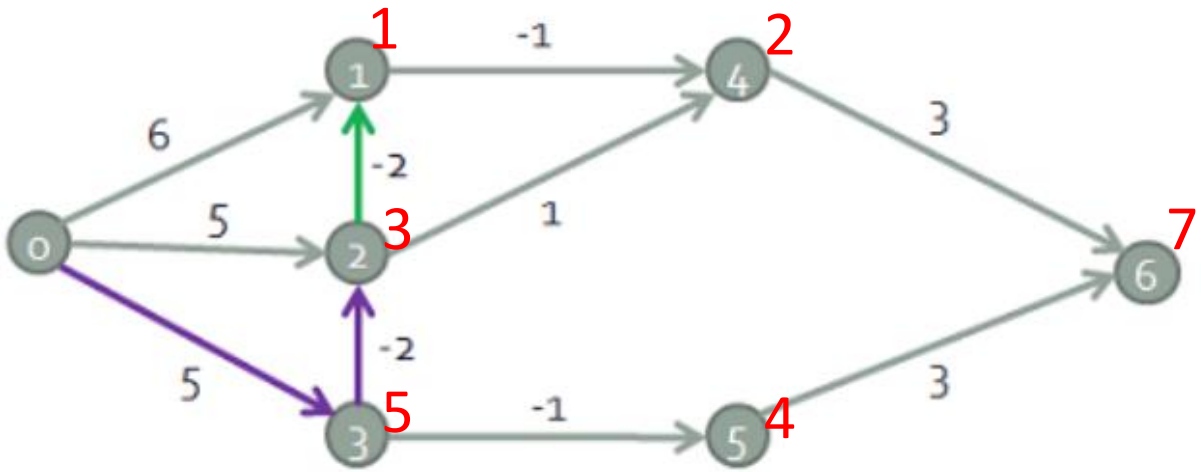
k	1	2	3	4	5	6
1	6	5	5	∞	∞	∞
2						
3						
4						
5						
6						

Example 2 Answer



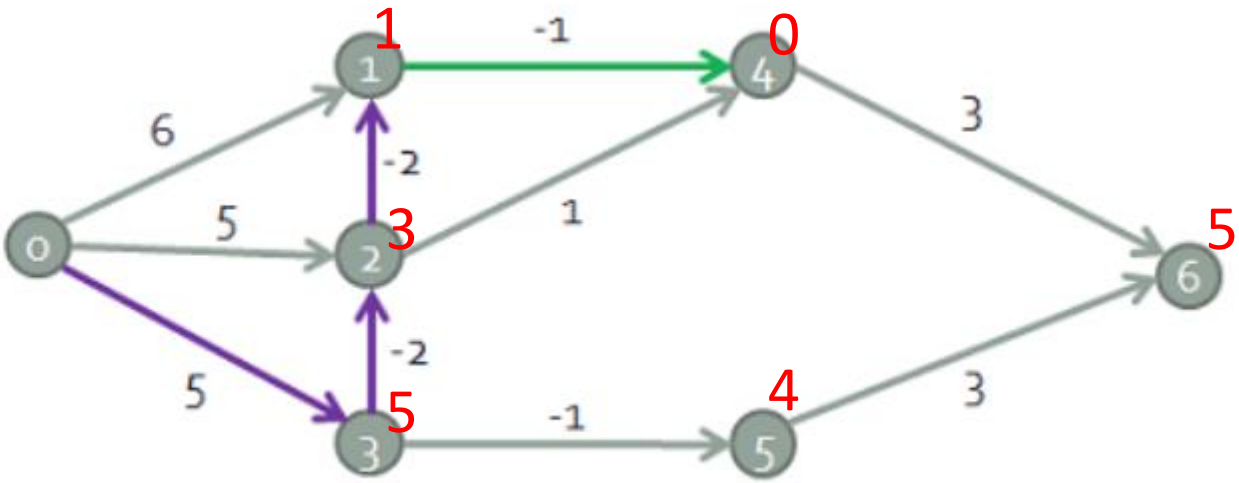
k	1	2	3	4	5	6
1	6	5	5	∞	∞	∞
2	3	3	5	5	4	∞
3						
4						
5						
6						

Example 2 Answer



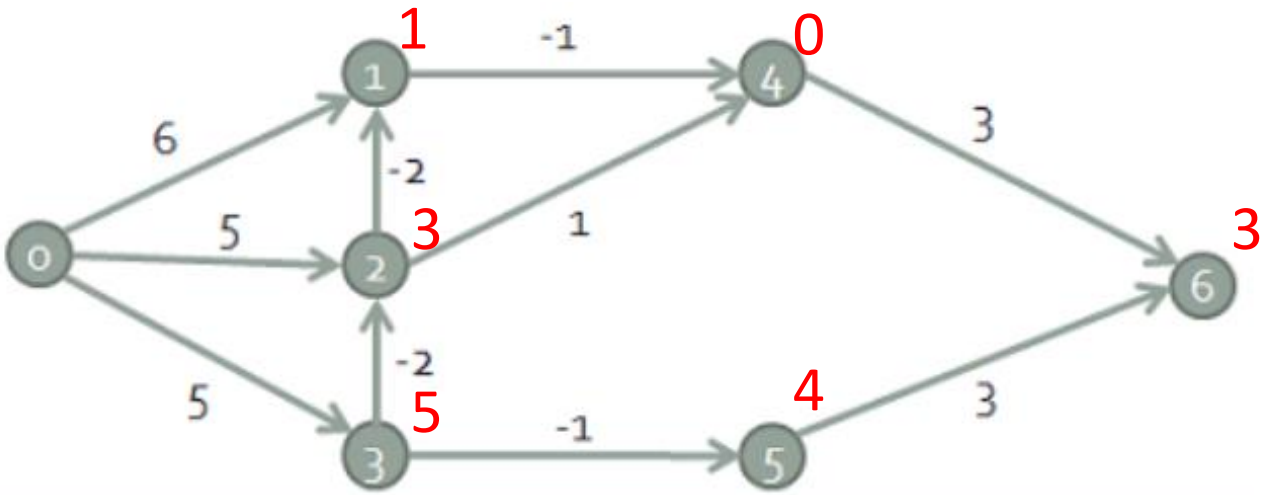
k	1	2	3	4	5	6
1	6	5	5	∞	∞	∞
2	3	3	5	5	4	∞
3	1	3	5	2	4	7
4						
5						
6						

Example 2 Answer



k	1	2	3	4	5	6
1	6	5	5	∞	∞	∞
2	3	3	5	5	4	∞
3	1	3	5	2	4	7
4	1	3	5	0	4	5
5						
6						

Example 2 Answer



k	1	2	3	4	5	6
1	6	5	5	∞	∞	∞
2	3	3	5	5	4	∞
3	1	3	5	2	4	7
4	1	3	5	0	4	5
5	1	3	5	0	4	3
6	1	3	5	0	4	3

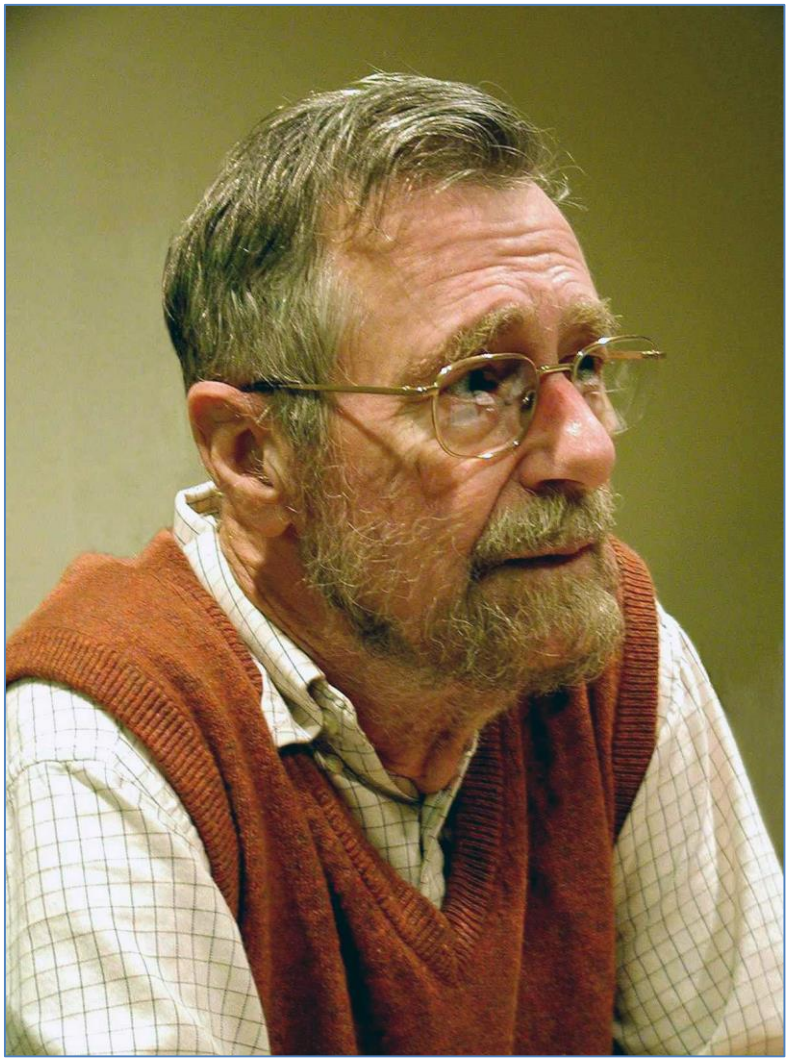
Summary Bellman-Ford algorithm

- The “value function” (weight of shortest path) at each node satisfies a set of Bellman equations.
- These are sets of equalities per node, and can be solved by iterations starting from some conservative estimate.
- A suitable iterative method is based on edge relaxations.
- If we could guess the shortest path tree, then $|V|-1$ relaxations could lead to the solution of the value function.
- We must be smart how we choose the relaxations because there can be exponentially many.
- Bellman-Ford algorithm: chooses relaxations in a “worst case” fashion and has complexity $\Theta(nm)$.

Dijkstra's shortest path algorithm

- Applies to graphs with only nonnegative weights;
- Maintains a set of vertices with established shortest distances;
- Promotes a vertex with minimal estimated distance to set of established vertices;
- Uses relaxation to improve estimated distances of non-established vertices via newly promoted vertex;
- Terminates when all vertexes have established (shortest) distances.

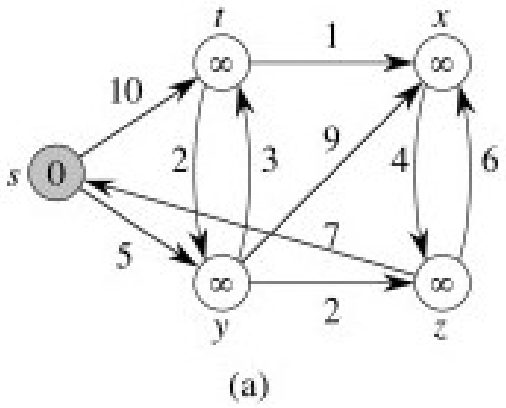
Edsger W. Dijkstra (1930-2002)



Computer Scientist with many foundational contributions to CS:

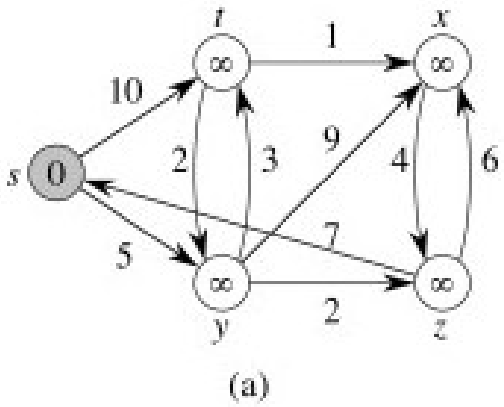
- recursion
- distributed programming
- operating systems
- structured programming
- algorithms
- programming language semantics
- correctness proofs
- etc. etc.

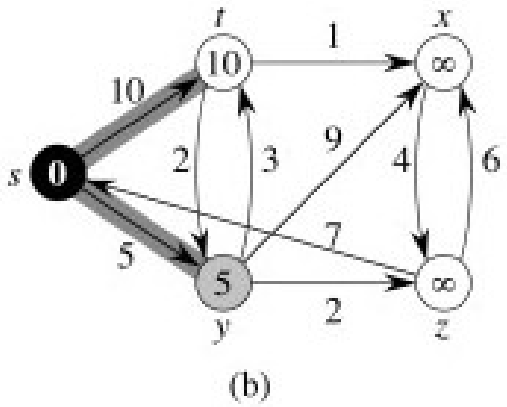
Example execution



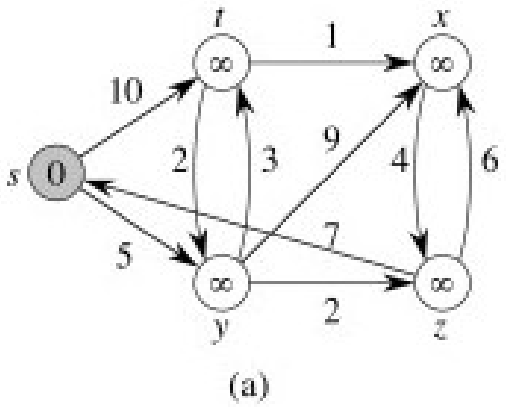
(a)

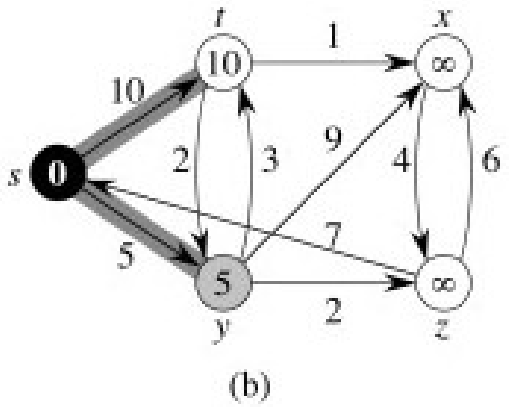
Example execution

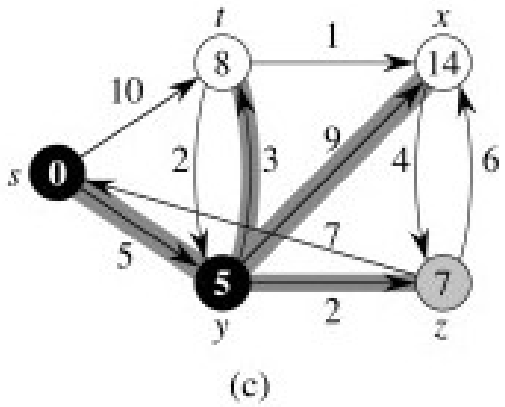




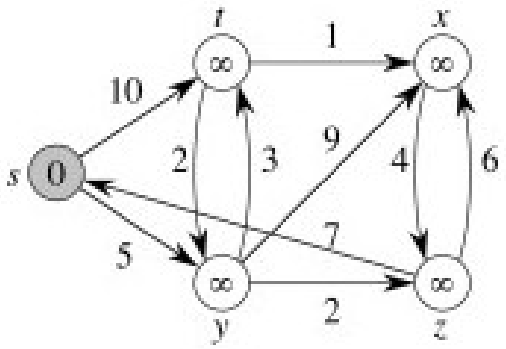
Example execution



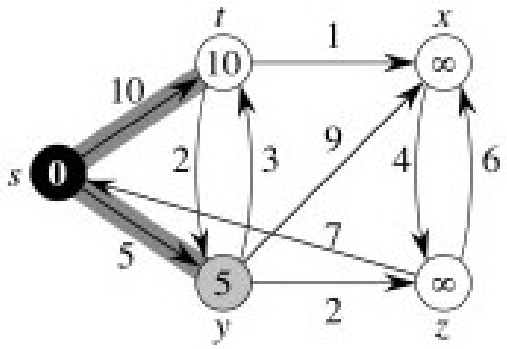




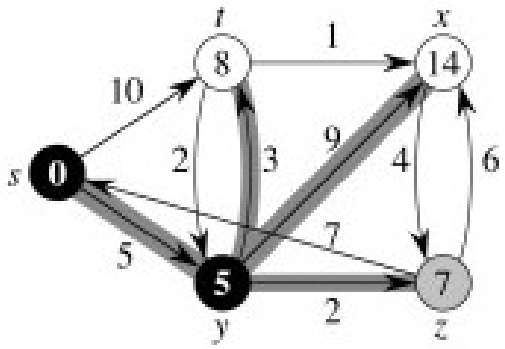
Example execution



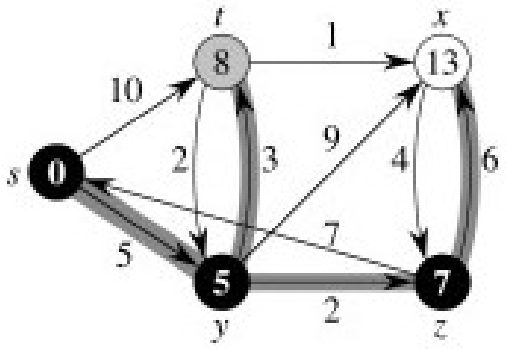
(a)



(b)

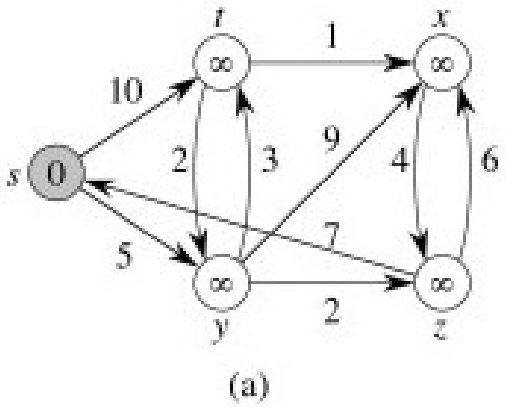


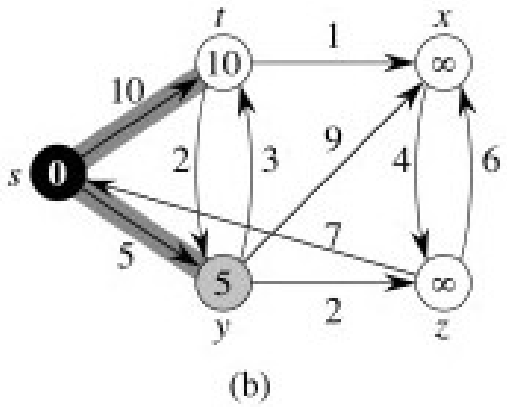
(c)

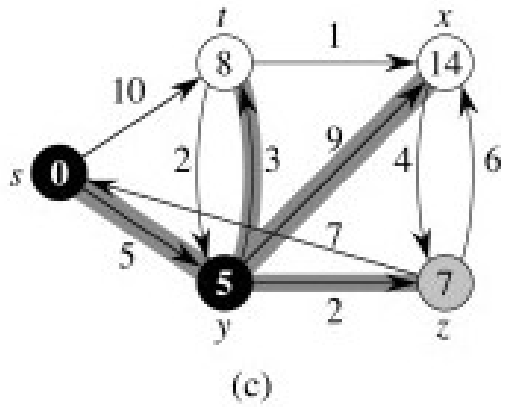


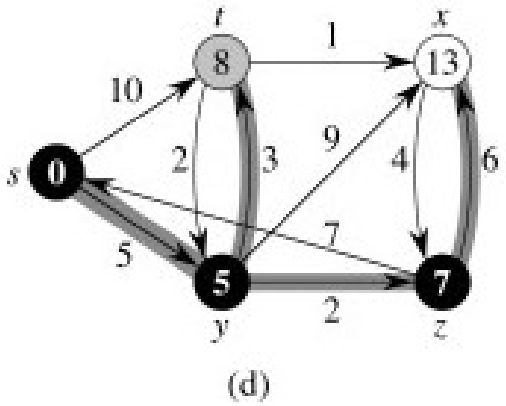
(d)

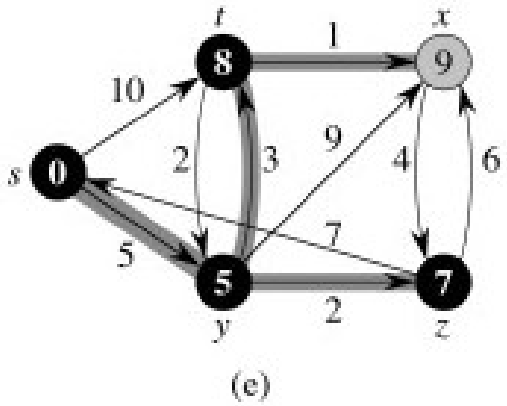
Example execution



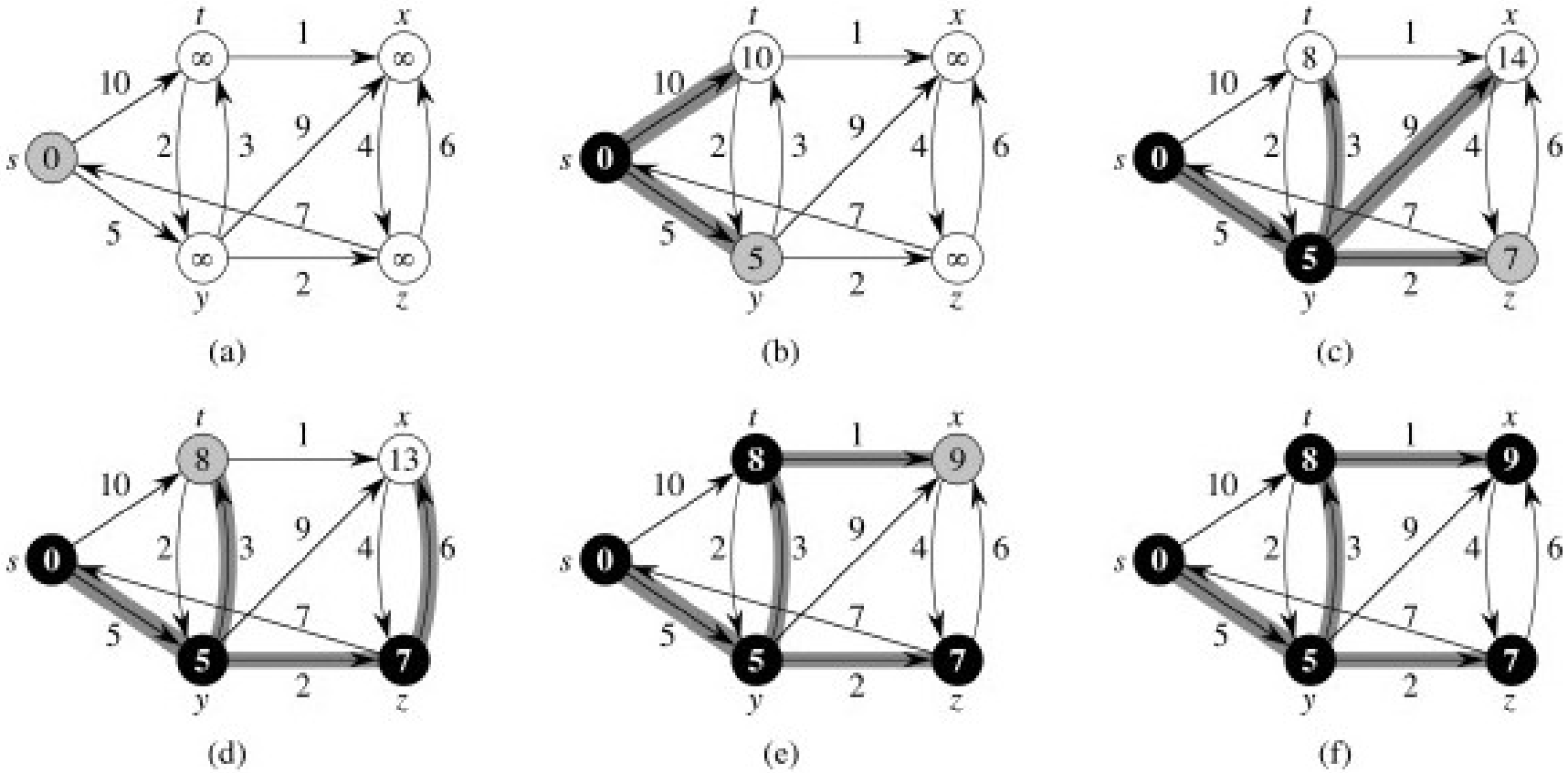








Example execution



Pseudo-code

for v in V:
    $v.d = \infty$
    $v.\pi$ = nil
$s.d = 0$
$S = \emptyset$

initialization

while $S \neq V$:
    $u = v$ with $v.d \leq w.d$ for all w not in S
    $S = S \cup \{u\}$
    for each edge (u, v) in E, v not in S:
        relax(u, v)

main loop

Complexity

for v in V:
    $v.d = \infty$
    $v.\pi$ = nil
$s.d = 0$
$S = \emptyset$

initialization

$\Theta(|V|)$

while $S \neq V$:
    $u = v$ with $v.d \leq w.d$ for all w not in S
    $S = S \cup \{u\}$
    for each edge (u, v) in E, v not in S:
        relax(u, v)

main loop

$O(|V|^2 + \lceil E \rceil) = O(|V|^2)$

Smart implementations using priority queues for storing and retrieving the values $v.d$ can improve complexity further, e.g. $O(|V| \log(|V|) + |E|)$, using a so-called Fibonacci heap.