COMP319 Algorithms 1
Lecture 15
Shortest Paths

Instructor: Gil-Jin Jang

School of Electronics Engineering, Kyungpook National University

Textbook Chapters 25 and 26

Slide credits: 홍석원, 명지대학교; 김한준, 서울시립대학교;
J. Lillis, UIC; Roger Crawfis, CSE 680;

George Bebis, Analysis of Algorithms, CS 477/677
David Luebke, CS332, Virginia University

Table of Contents

Slide credits: 홍석원; 김한준; J. Lillis; Roger Crawfis; George Bebis; David Luebke

Shortest Paths

- Precondition
  - Given directed graph with edge weights
    - Undirected graphs can be translated to directed graphs with edges in both direction with the same weights
    - Undirected edge (*u*, *v*; *w*) implies directed edges (*u*, *v*; *w*) and (*v*, *u*; *w*)
- Shortest path between two vertices
  - the path that has the minimum sum of the edge weights among all possible paths
  - If there is a cycle with its sum of edge weights is negative, the problem is unsolvable
    - End up with infinite loop

Shortest Path Problems

- How can we find the shortest route between two points on a road map?

- Model the problem as a graph problem:
  - Road map is a weighted graph:

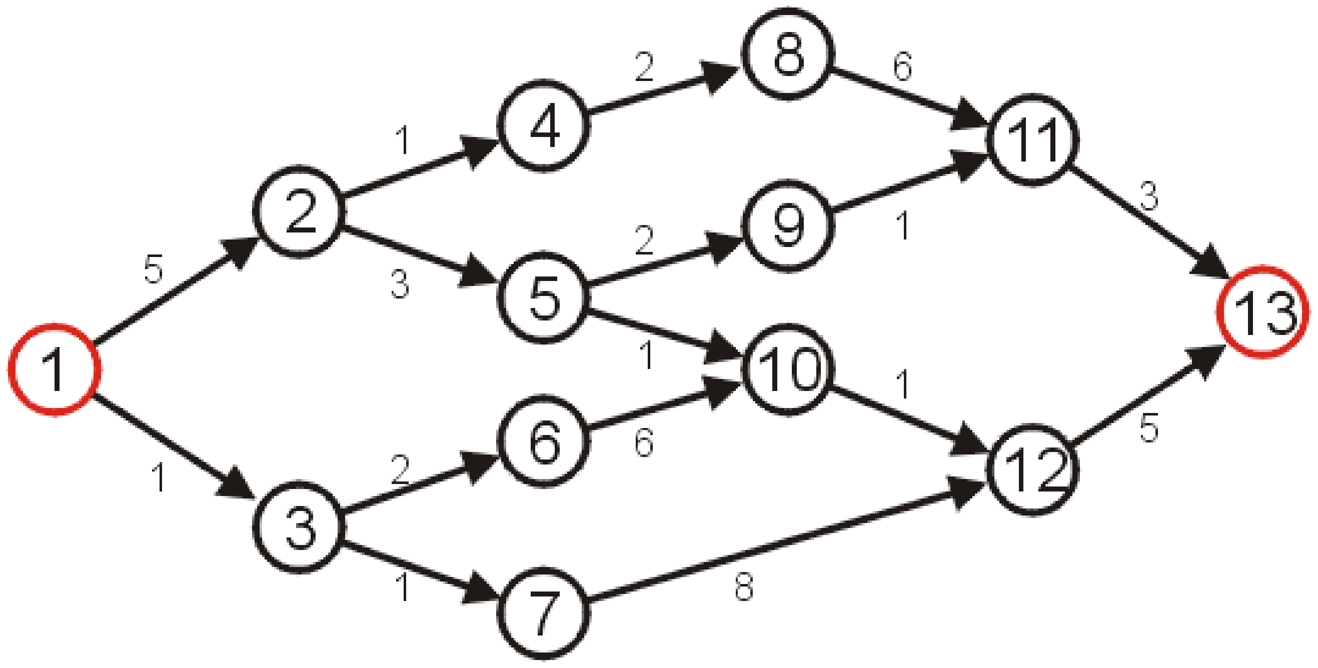
    vertices = cities

    edges = road segments between cities

    edge weights = road distances
  - Goal: find a shortest path between two vertices (cities)

Shortest Path

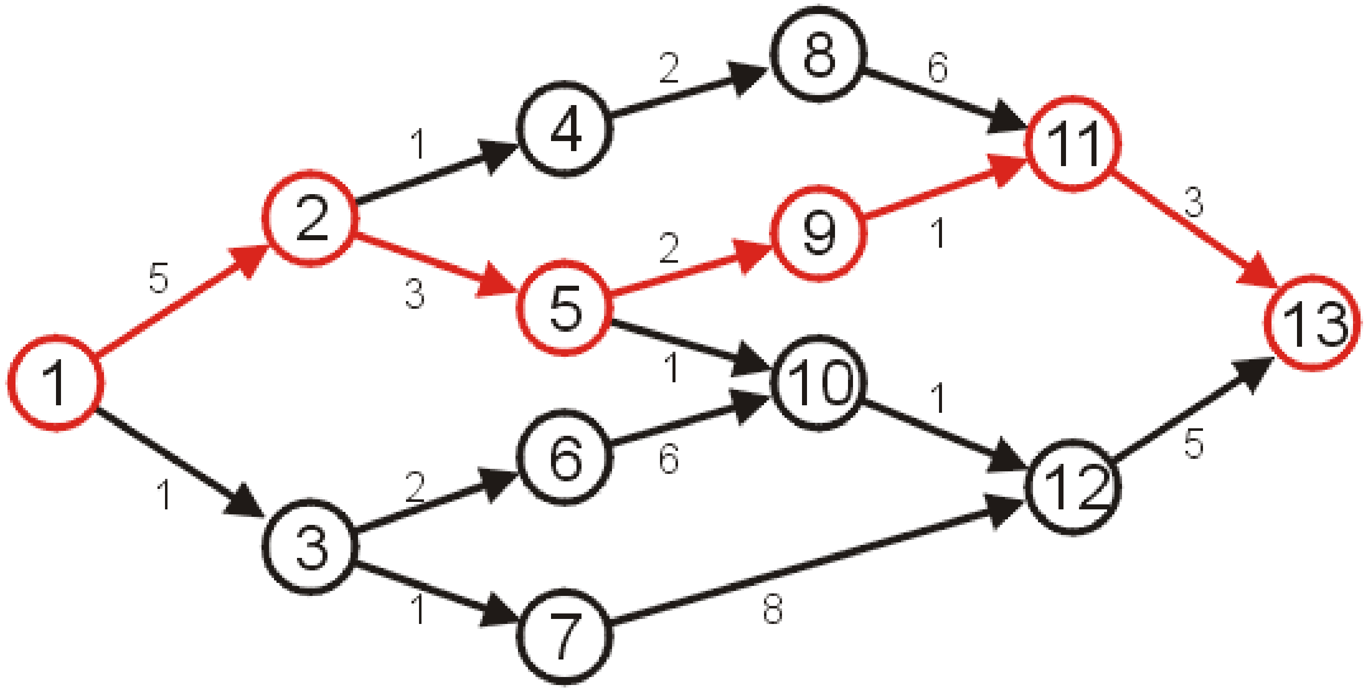
- Given the graph below, suppose we wish to find the shortest path from vertex 1 to vertex 13



$$\begin{tikzcd}[column sep=large]
 & & & 8 \arrow[dr, "6"] & & \\
 & & 4 \arrow[ur, "2"] & & 11 \arrow[dr, "3"] & \\
 & 2 \arrow[ur, "1"] \arrow[r, "3"] & 5 \arrow[r, "2"] \arrow[dr, "1"] & 9 \arrow[ur, "1"] & & 13 \\
1 \arrow[ur, "5"] \arrow[dr, "1"] & & & 10 \arrow[r, "1"] & 12 \arrow[ur, "5"] & \\
 & 3 \arrow[r, "2"] \arrow[dr, "1"] & 6 \arrow[ur, "6"] & & & \\
 & & 7 \arrow[uurr, "8"] & & &
\end{tikzcd}$$

Shortest Path

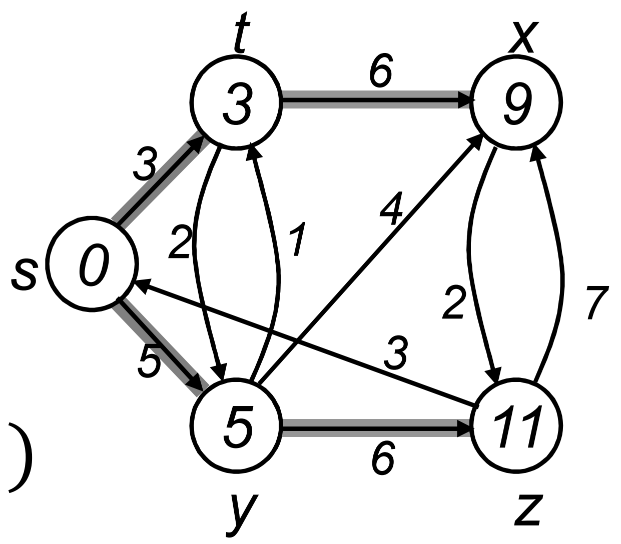
- After some consideration, we may determine that the shortest path is as follows, with length 14
  - Other paths exists, but they are longer



Formulation of Shortest Path Problem

- Input:
  - Directed graph *G* = (*V*, *E*)
  - Weight function *w*: *E* → *R*
- Weight of path $\mathbf{p} = \langle v_0, v_1, \cdots v_k \rangle$

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



- Shortest-path weight from u to v:

$$\delta(u, v) = \begin{cases} w(\mathbf{p}) : u \overset{\mathbf{p}}{\rightarrow} v & \textit{there exists a path from } u \textit{ to } v \\ \infty & \textit{otherwise} \end{cases}$$

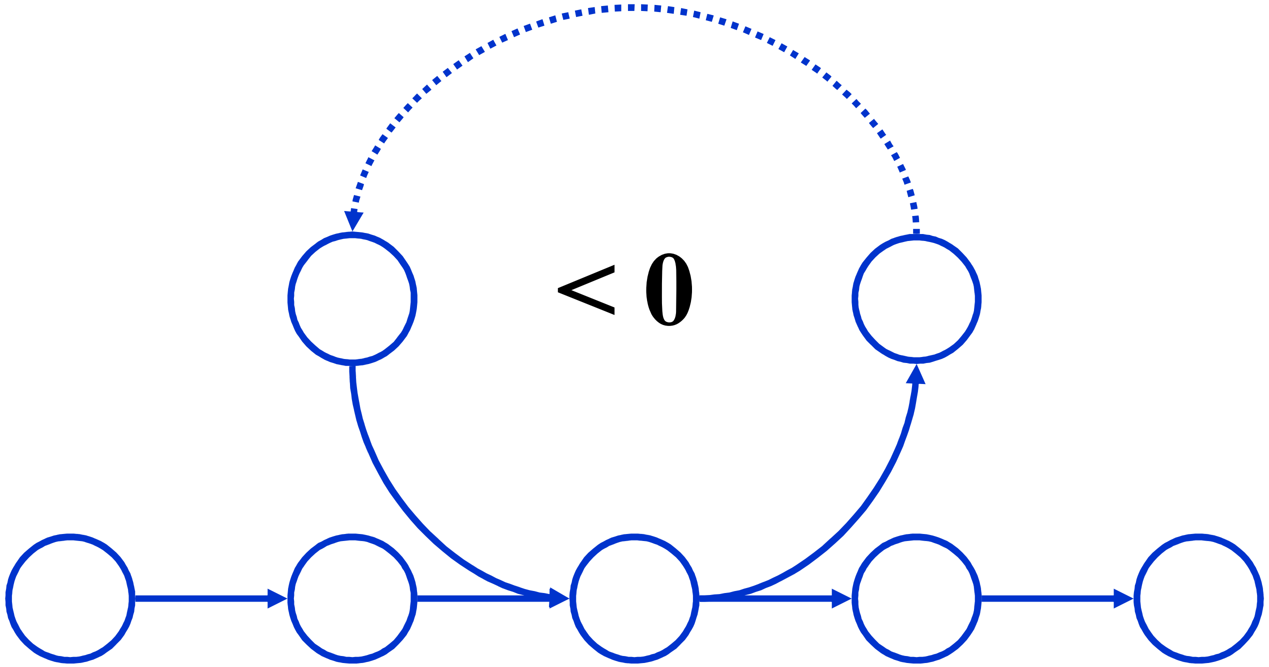
- **Note**:
  - there might be multiple shortest paths from u to v

Discussion Items

- How many possible paths are there from *s* to *t*?
- Any suggestions on how to reduce the set of possibilities?
- Can we safely ignore cycles? If so, how?
- Can we determine a lower bound on the complexity like we did for comparison sorting?

Shortest Path Properties

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

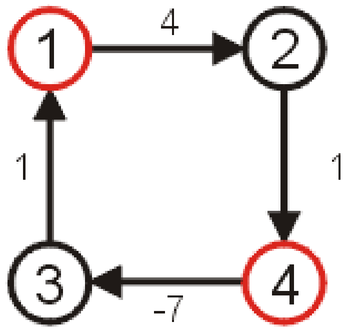


(continued)

Slide credits: 홍석원; 김한준; J. Lillis; Roger Crawfis; George Bebis; David Luebke

Negative Cycles

- Clearly, if we have negative vertices, it may be possible to end up in a cycle whereby each pass through the cycle decreases the total length
- Thus, a shortest length would be undefined for such a graph
- Consider the shortest path from vertex 1 to 4...
- We will only consider non-negative weights.



More on Cycles

- Can shortest paths contain cycles?

- Negative-weight cycle: *No*
  - Shortest path is not well defined
- Positive-weight cycles: *No*
  - By removing the cycle, we can get a shorter path
- Zero-weight cycles: *Useless*
  - No reason to use them
  - Can remove them to obtain a path with the same weight

Variants of Shortest Path

- **Single-source** shortest paths
  - $G = (V, E) \Rightarrow$ find a shortest path from a given source vertex s to each vertex $v \in V$
- **Single-destination** shortest paths
  - Find a shortest path to a given destination vertex t from each vertex v
  - Reversing the direction of each edge $\Rightarrow$ single-source
- **Single-pair** shortest path
  - Find a shortest path from u to v for given vertices u and v
- **All-pairs** shortest paths
  - Find a shortest path from u to v for every pair of vertices, (u, v)

SINGLE-SOURCE SHORTEST PATH

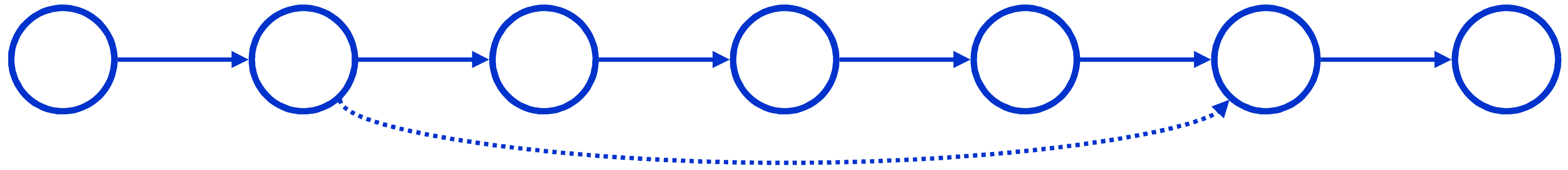
Slide credits: 홍석원; 김한준; J. Lillis; Roger Crawfis; George Bebis; David Luebke

Single-Source Shortest Path

- Problem: given a weighted directed graph *G*, find the minimum-weight path from a given source vertex *s* to another vertex *v*
  - "Shortest-path" = minimum weight
  - Weight of path is sum of edges
  - E.g., a road map: what is the shortest path from Chapel Hill to Charlottesville?

Shortest Path Properties

- We have *optimal substructure*: the shortest path consists of shortest subpaths:



Proof: Assume that p_{ij} is a shortest path from v_i to v_j

$$p = v_1 \overset{p_{1i}}{\rightsquigarrow} v_i \overset{p_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$$

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

Assume $\exists\, p_{ij}'$ from v_i to v_j with $w(p_{ij}') < w(p_{ij})$

$\Rightarrow w(p') = w(p_{1i}) + w(p_{ij}') + w(p_{jk}) < w(p)$ contradiction!

Algorithm Overview

- Bellman-Ford algorithm
  - Negative weights are allowed
  - Negative cycles reachable from the source are not allowed.
- Dijkstra’s algorithm
  - Negative weights are not allowed
- Operations common in both algorithms:
  - Initialization
  - Relaxation

Notation

For each vertex $v \in V$:

- $\delta(s, v)$: **shortest-path weight**
- d[v]: shortest-path weight **estimate** (or **upperbound** found so far)
  - Initially, $d[v]=\infty$
  - $d[v] \rightarrow \delta(s,v)$ as algorithm progresses
- $\pi[v]$ = **predecessor** of v on a shortest path from ***s***
  - If no predecessor, $\pi[v]$ = NIL
  - π induces a tree—**shortest-path tree**

Slide credits: 홍석원; 김한준; J. Lillis; Roger Crawfis; George Bebis; David Luebke

Initialization

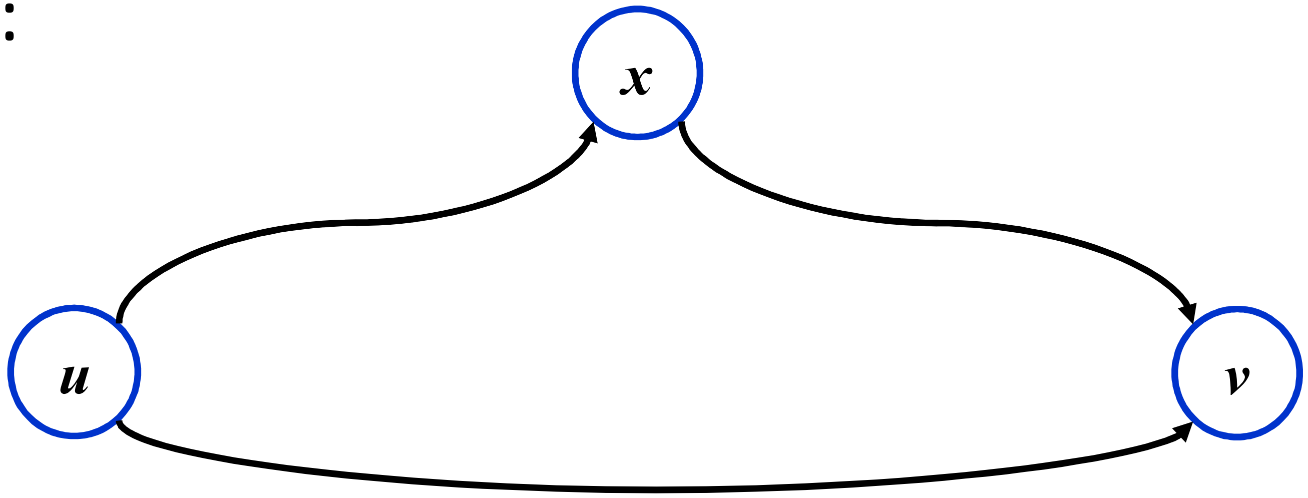
Alg.: INITIALIZE-SINGLE-SOURCE(V, s)

```
1.  for each $v \in V$
2.       do $d[v] \leftarrow \infty$
3.          $\pi[v] \leftarrow$ NIL
4.  $d[s] \leftarrow 0$
```

- All the shortest-paths algorithms start with INITIALIZE-SINGLE-SOURCE

Triangular Inequality

- Define $\delta(u,v)$ to be the weight of the shortest path from u to v
- Shortest paths satisfy the *triangular inequality*:
  - $\delta(u,v) \leq \delta(u,x) + \delta(x,v)$
- "Proof":



This path is no longer than any other path

Relaxation

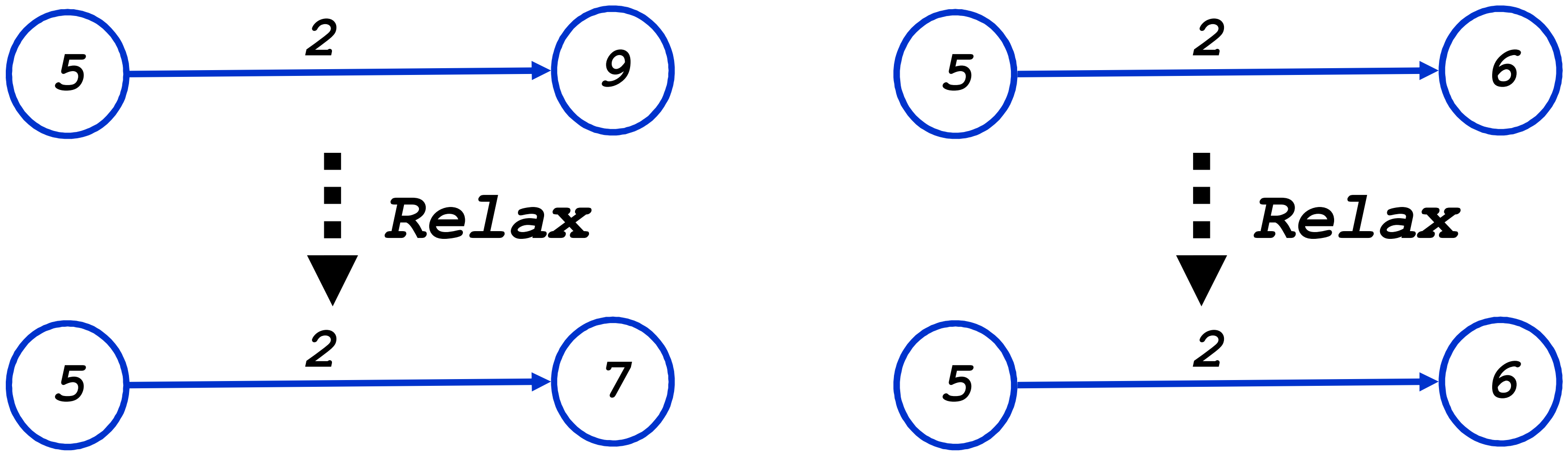
- A key technique in shortest path is *relaxation*
  - Idea: relaxing an edge (u, v) = testing whether we can improve the shortest path to v found so far by going through u
  - for all v, maintain upper bound d[v] on $\delta(s,v)$

```
Relax(u,v,w) {
    if ( d[v] > d[u]+w(u,v) ) then
      // we can improve the shortest path to v
      d[v] = d[u]+w(u,v)
      π[v] = u
}
```

After relaxation:
$d[v] \leq d[u] + w(u, v)$

Relaxation Example

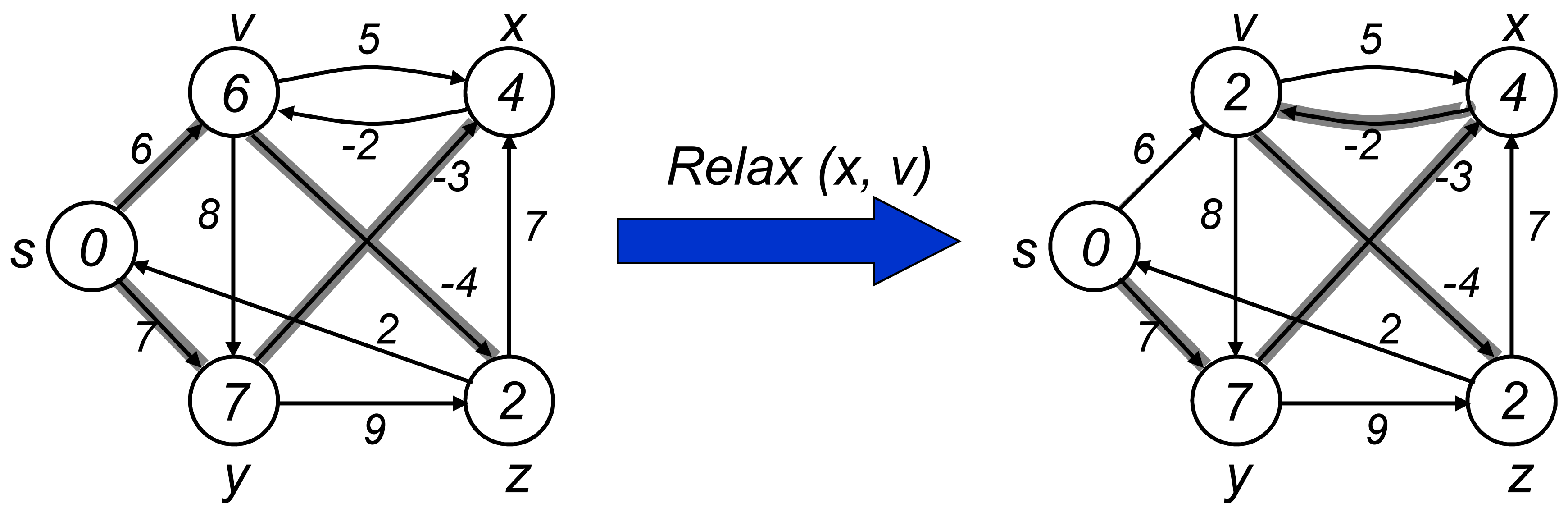
```
Relax(u,v,w) {
    if ( d[v] > d[u]+w(u,v) ) then
      d[v] = d[u]+w(u,v)
      π[v] = u
}
```



Slide credits: 홍석원; 김한준; J. Lillis; Roger Crawfis; George Bebis; David Luebke

Upper-bound property

- Upper-bound property by relaxation
  - We always have d[*v*] ≥ δ (*s*, *v*) for all *v*.
  - The estimate never goes up – relaxation only lowers the estimate



Single-source shortest path

BELLMAN-FORD ALGORITHM

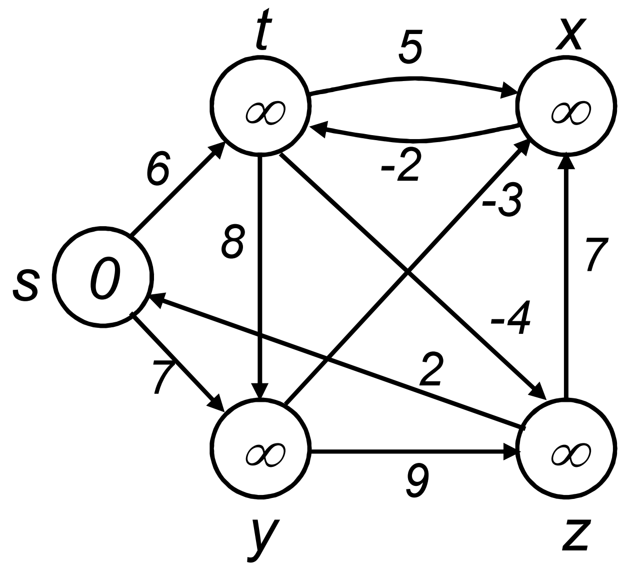
Bellman-Ford Algorithm

- Single-source shortest path problem
  - Computes $\delta(s, v)$ and $\pi[v]$ for all $v \in V$
- Allows negative edge weights - can detect negative cycles.
  - Returns TRUE if no negative-weight cycles are reachable from the source s
  - Returns FALSE otherwise $\Rightarrow$ no solution exists

Main Idea of Bellman-Ford

- Each edge is relaxed |V–1| times by making |V-1| passes over the whole edge set.
  - To make sure that each edge is relaxed exactly |V – 1| times, it puts the edges in an unordered list and goes over the list |V – 1| times.

(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)



Bellman-Ford by Dynamic Programming

- d_t^k : shortest distance from vertex r to t with at most k edges along the path
- Goal: find d_t^{n-1}

✓ *Recurrence relation*

$$\begin{cases} d_v^k = \min\limits_{\text{for all } (u,v)} \{d_u^{k-1} + w_{u,v}\}, & k > 0 \\ d_r^0 = 0 \\ d_t^0 = \infty, & t \neq r \end{cases}$$

Slide credits: 홍석원; 김한준; J. Lillis; Roger Crawfis; George Bebis; David Luebke

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

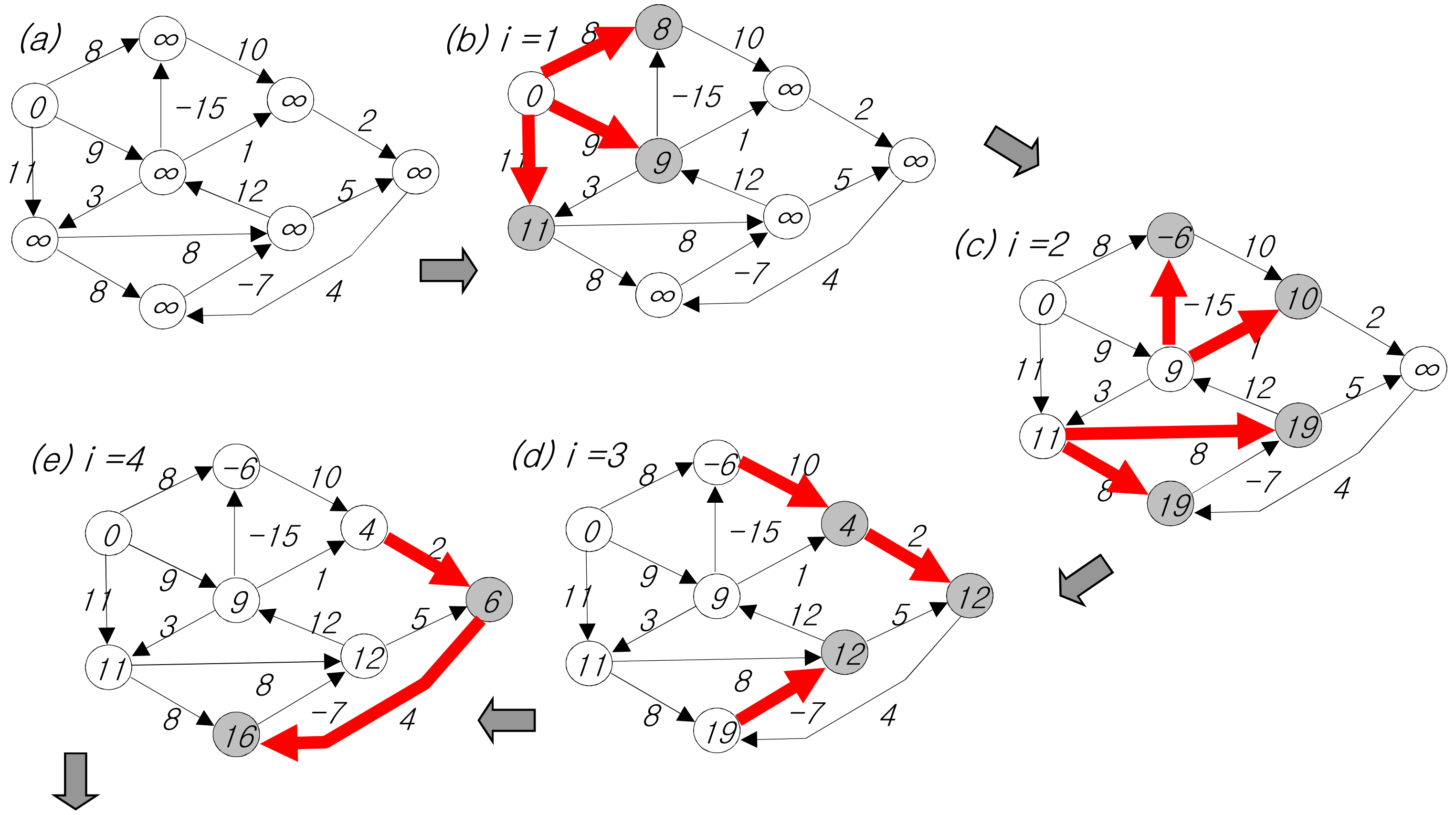
Relaxation: Make |V|-1 passes, relaxing each edge

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
```

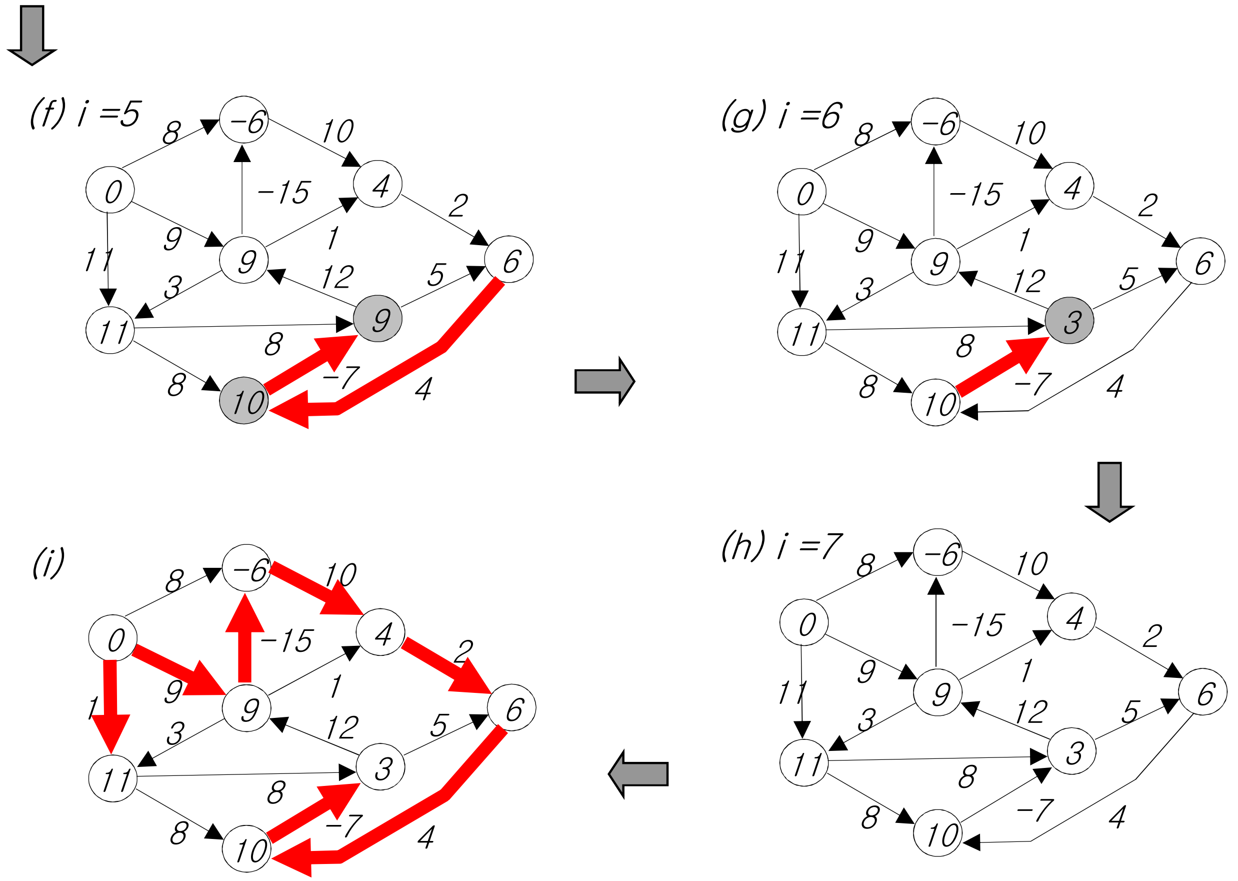
Slide credits: 홍석원; 김한준; J. Lillis; Roger Crawfis; George Bebis; David Luebke

Bellman-Ford Example



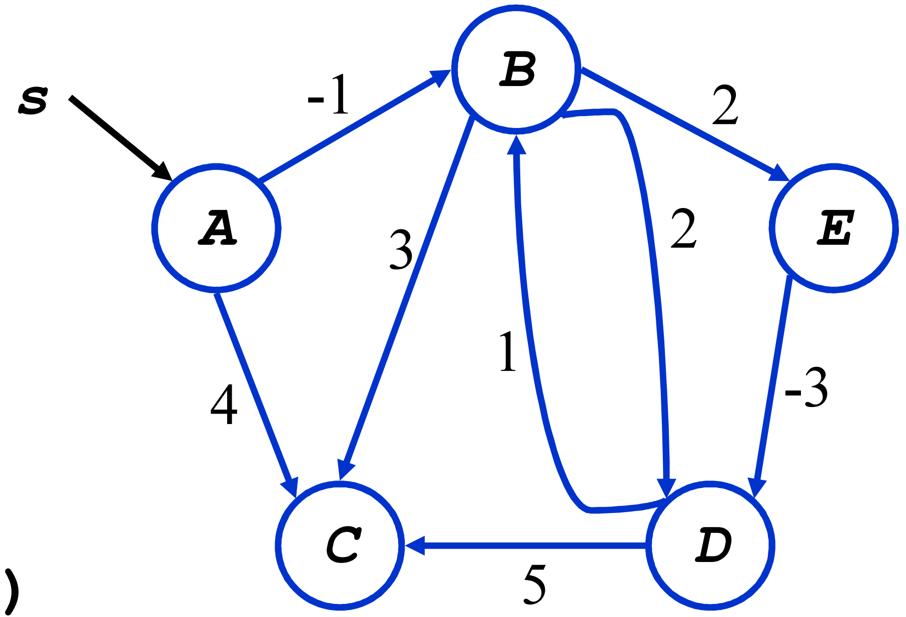
Slide credits: 홍석원; 김한준; J. Lillis; Roger Crawfis; George Bebis; David Luebke

Bellman-Ford Example



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";
```



Ex: work on board

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

Slide credits: 홍석원; 김한준; J. Lillis; Roger Crawfis; George Bebis; David Luebke

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?

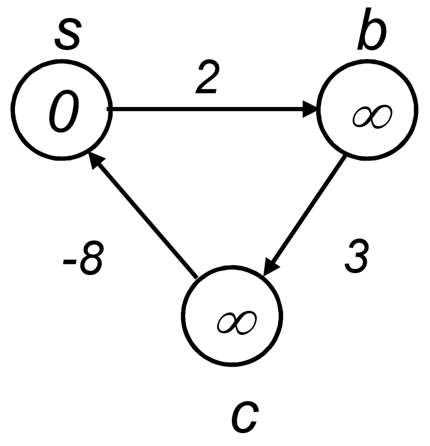
A: O(VE)

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

Slide credits: 홍석원; 김한준; J. Lillis; Roger Crawfis; George Bebis; David Luebke

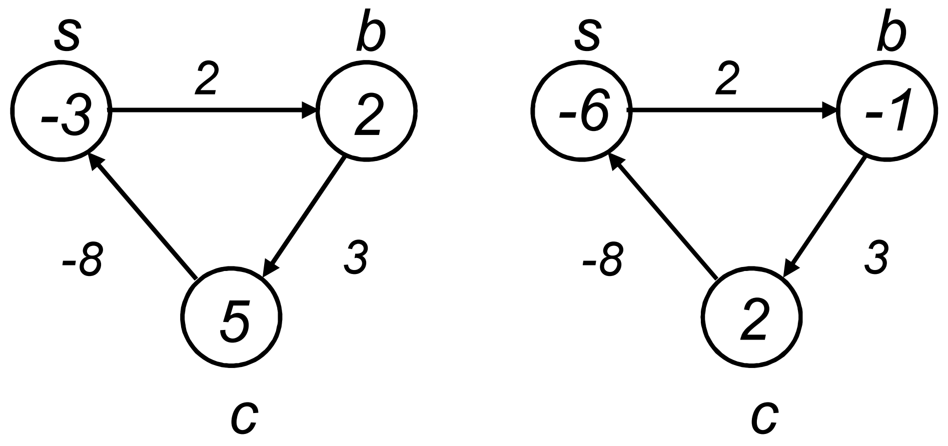
Detecting Negative Cycles (perform extra test after V-1 iterations)

```
for each edge $(u, v) \in E$
    do if $d[v] > d[u] + w(u, v)$
        then return FALSE
return TRUE
```



1st pass

2nd pass



(s,b) (b,c) (c,s)

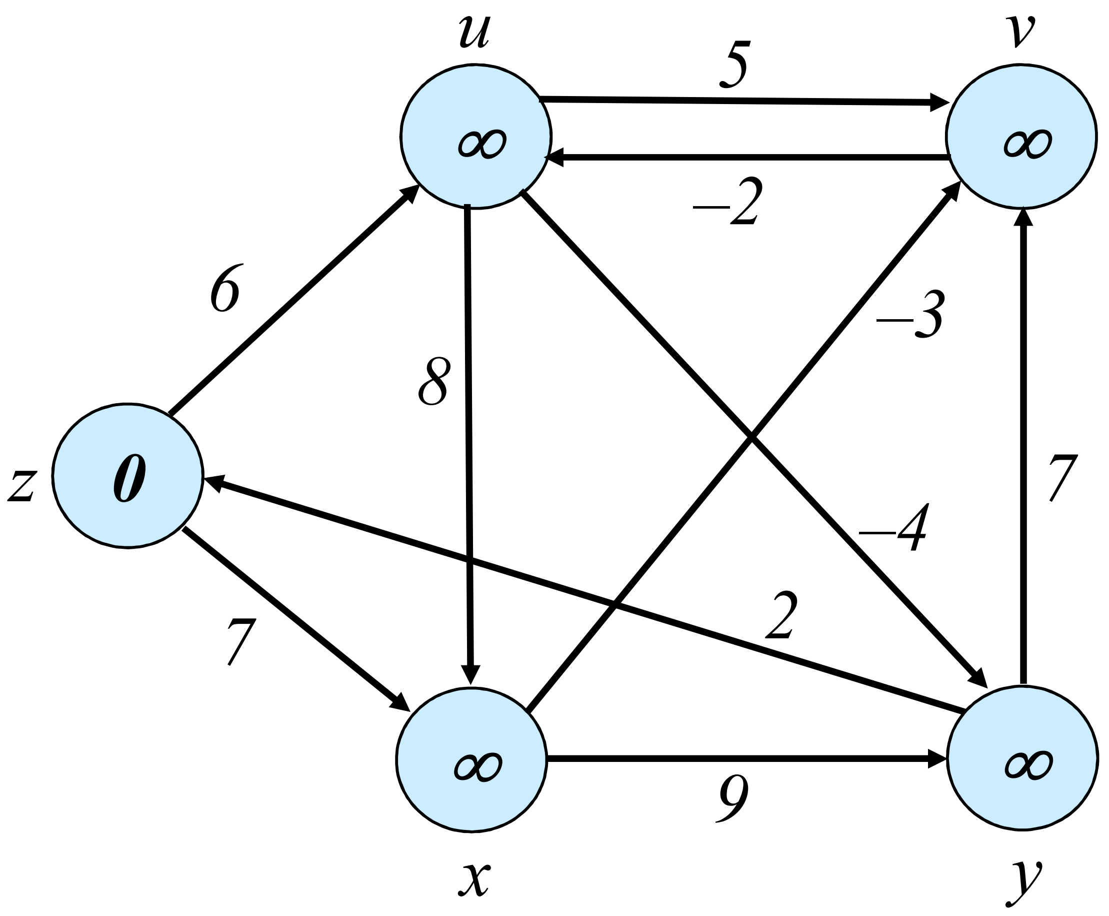
Look at edge (s, b):

$d[b] = -1$

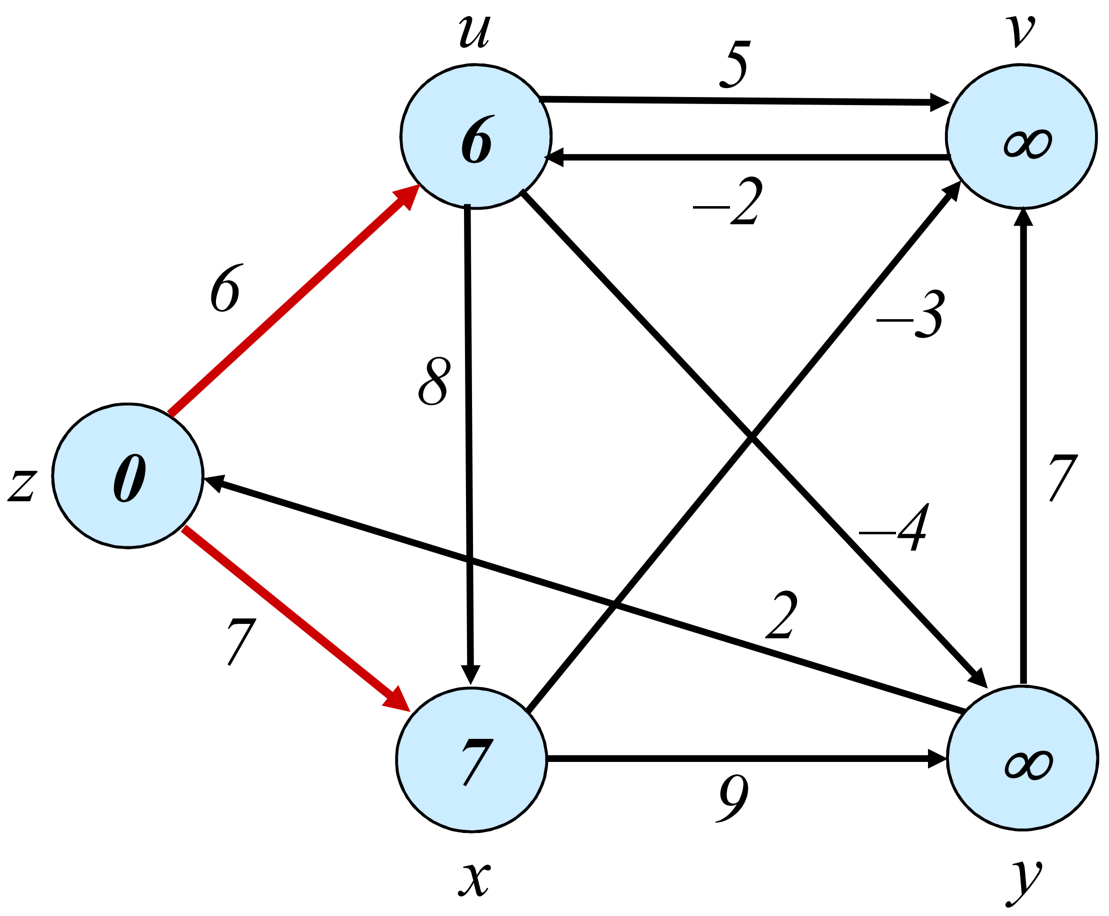
$d[s] + w(s, b) = -4$

$\Rightarrow d[b] > d[s] + w(s, b)$

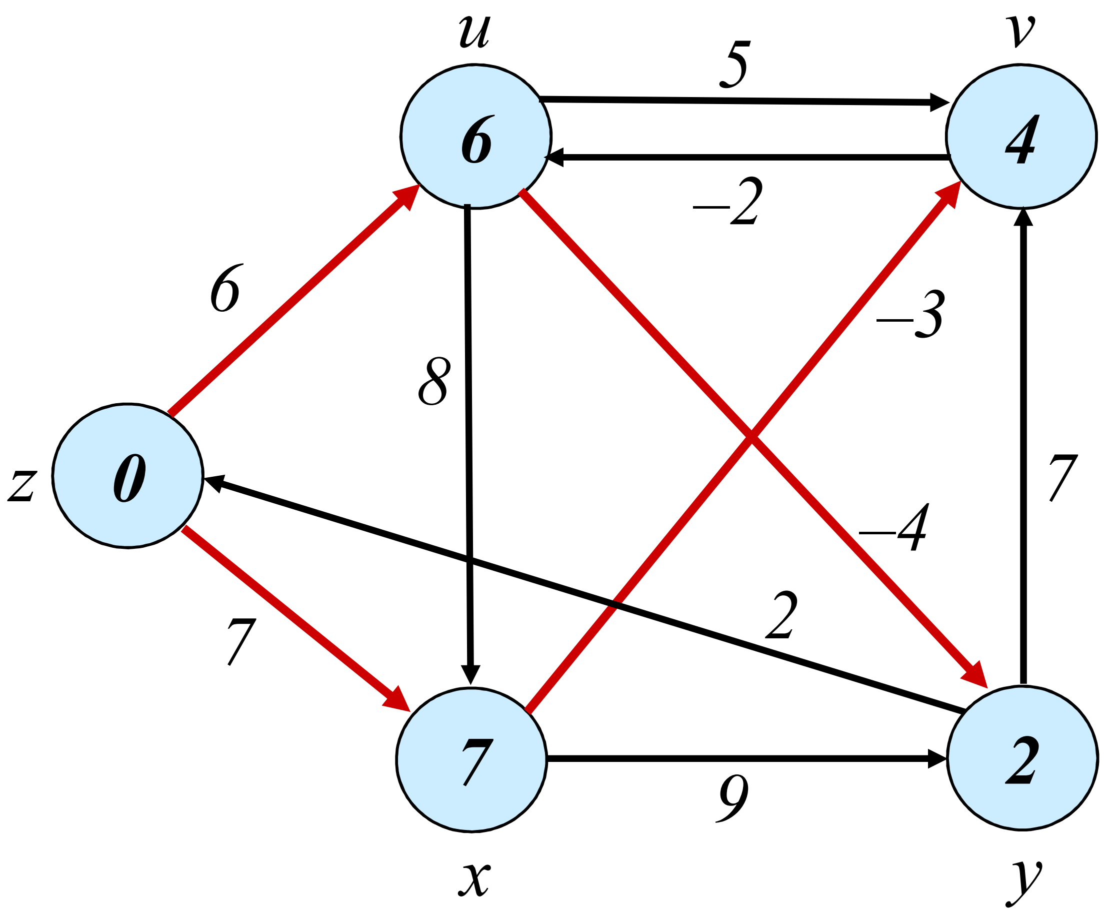
Example



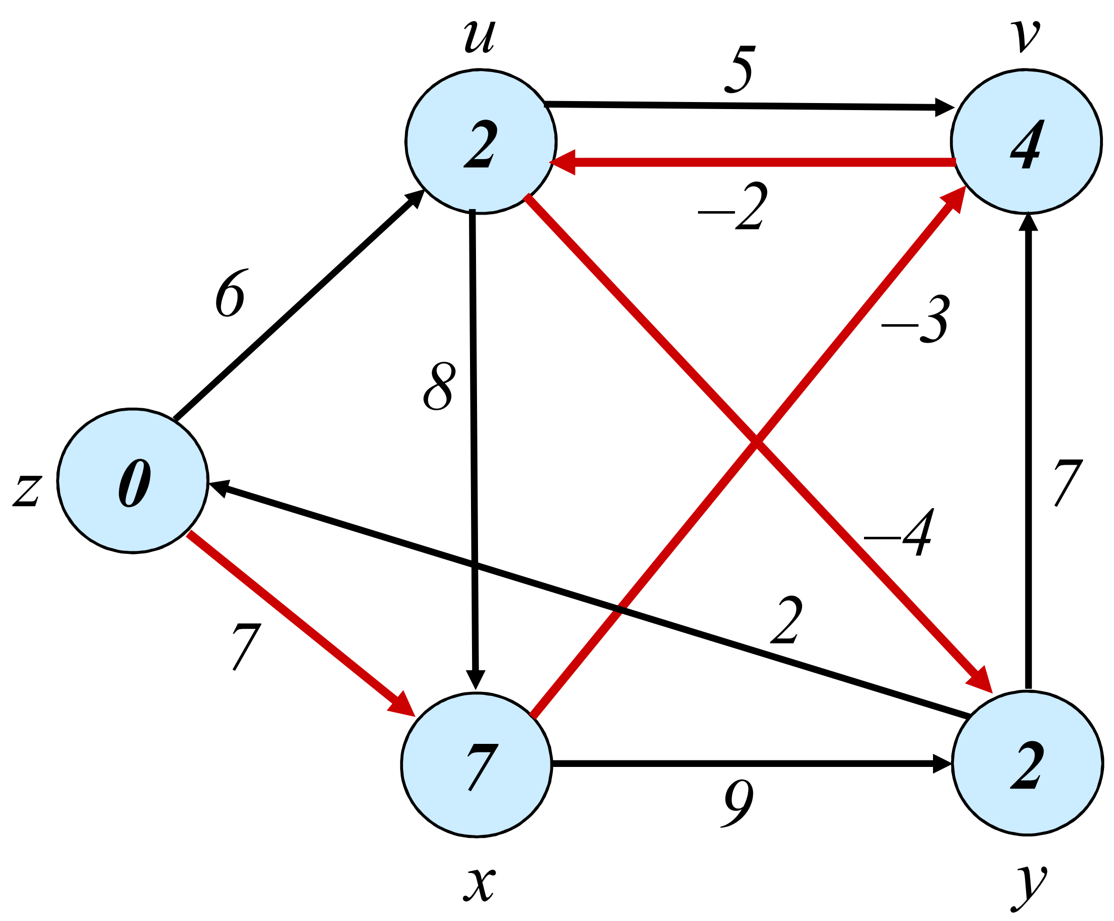
Example



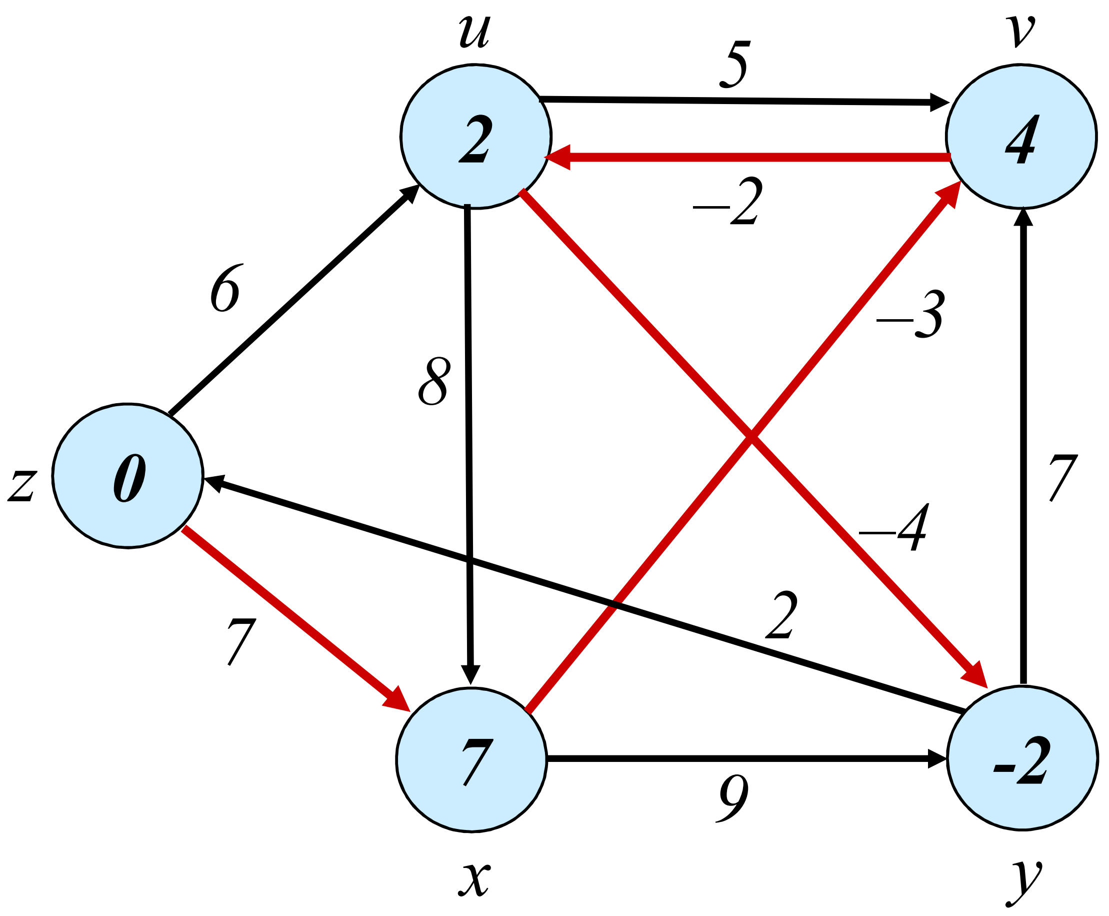
Example



Example



Example



Correctness of Bellman-Ford

- If $G=(V,E)$ contains no negative-weight cycles, then after then after the Bellman-Ford algorithm executes, $d[v] = \delta(s,v)$ for all $v \in V$.
- Proof. Let $v \in V$ be any vertex, and consider a shortest path ***p*** from *s* to *v* with the minimum number of edges.

$$s \to v_1 \to v_2 \to v_3 \to v_4 \to v$$

  - Since ***p*** is a shortest path, we have

$$\delta(s,v_i) = \delta(s,v_{i-1}) + w(v_{i-1}, v_i)\ .$$

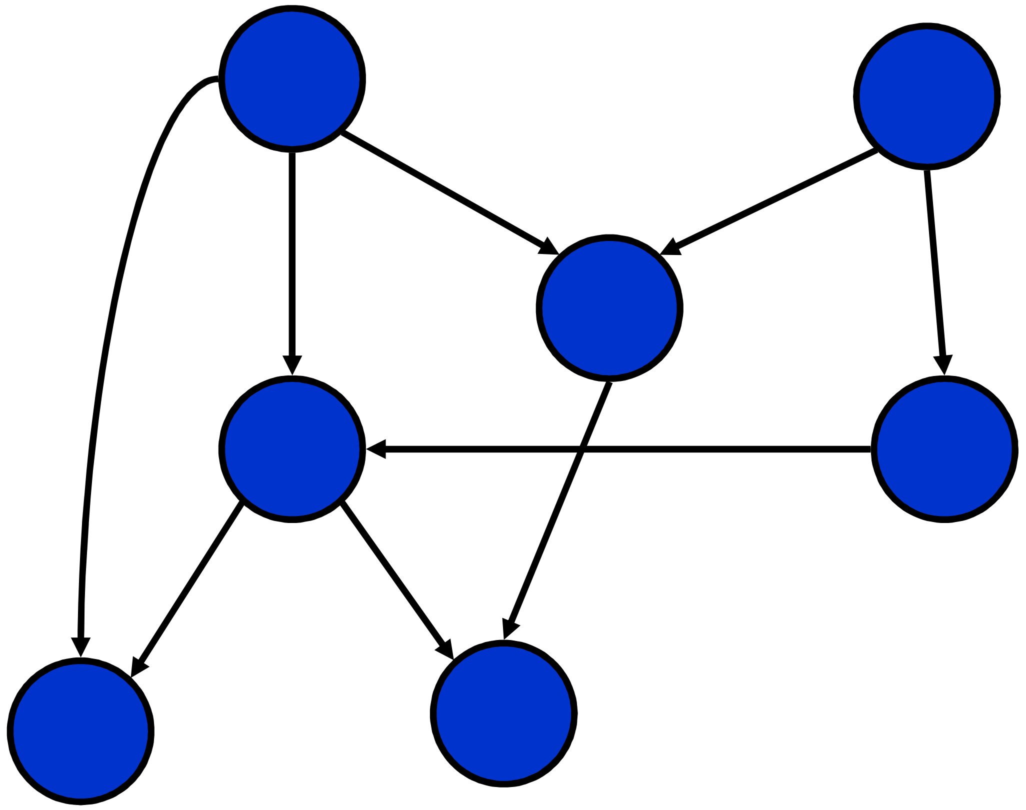
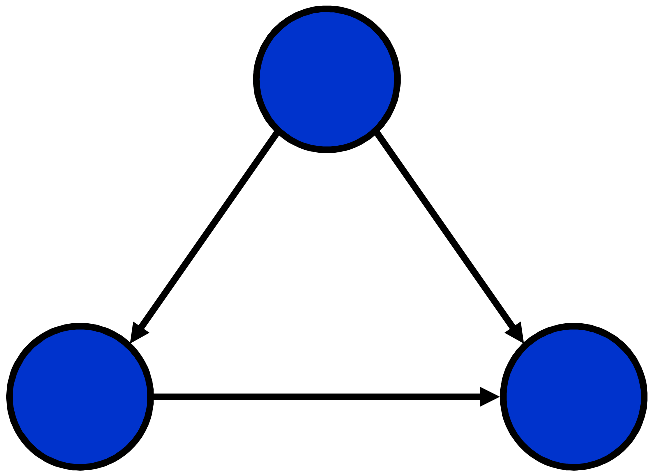
Correctness of Bellman-Ford

$$s \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v$$

- Initially, $d(v_0) = 0 = \delta(s,v_0)$, and $d[s]$ is unchanged by subsequent relaxations (because $d[v] \geq \delta(s,v)$).
  - After 1 pass through E, we have $d(v_1) = \delta(s,v_1)$.
  - After 2 passes through E, we have $d(v_2) = \delta(s,v_2)$.
  - …
  - After k passes through E, we have $d(v_k) = \delta(s,v_k)$.
- Since G contains no negative-weight cycles, ***p*** is a longest simple path that has $\leq |V| - 1$ edges.

Directed Acyclic Graphs

- A *directed acyclic graph* or *DAG* is a directed graph with no directed cycles:

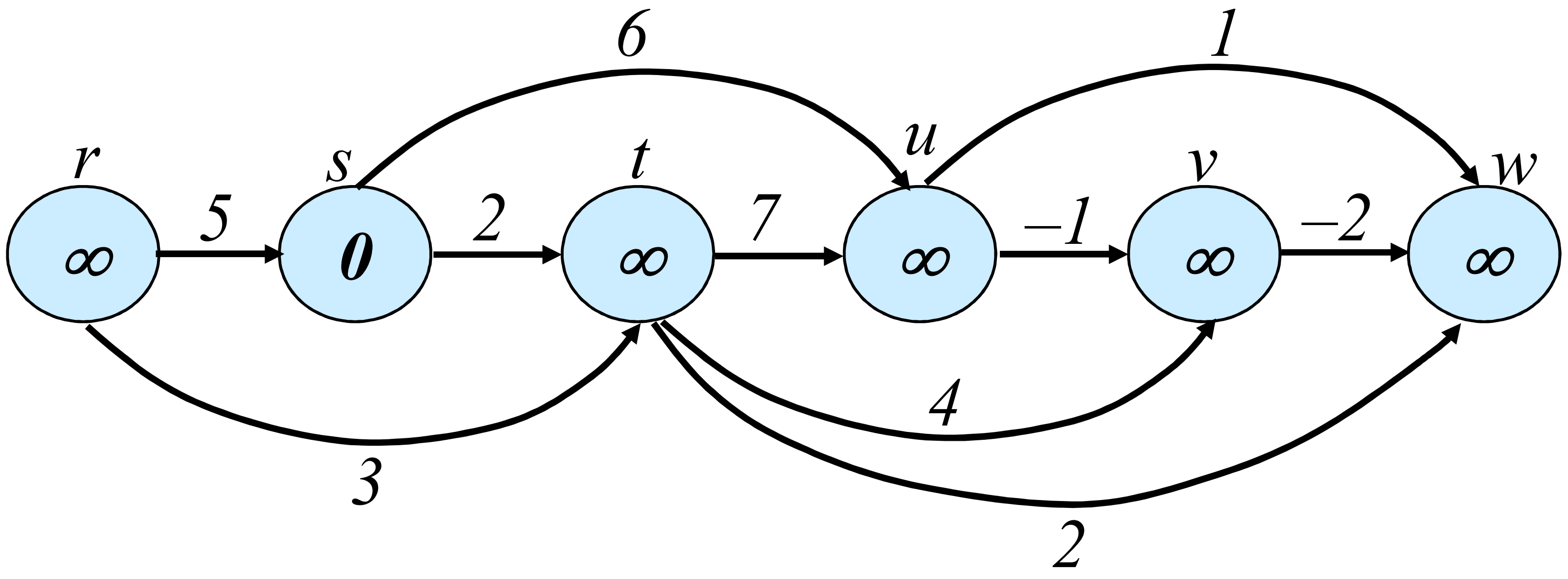



Slide credits: 홍석원; 김한준; J. Lillis; Roger Crawfis; George Bebis; David Luebke

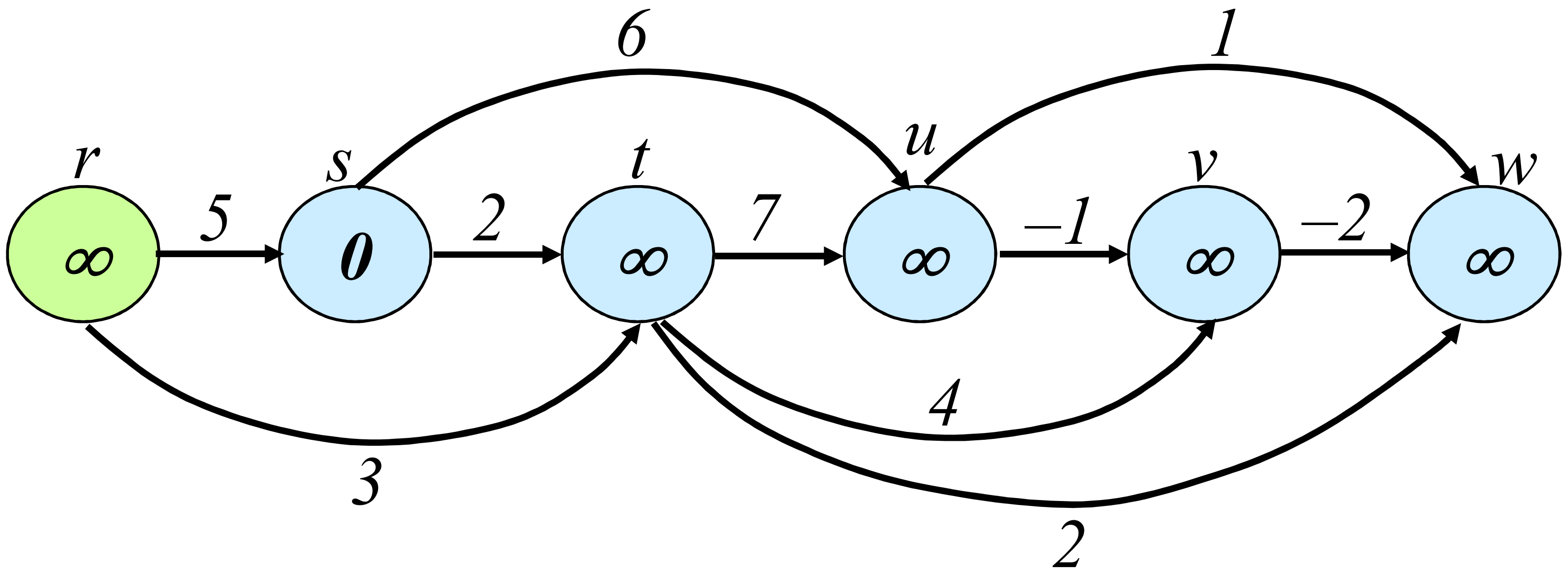
DAG Shortest Paths

- Problem: finding shortest paths in DAG
  - Bellman-Ford takes O(VE) time.
  - *How can we do better?*
  - Idea: use topological sort
    - If were lucky and processes vertices on each shortest path from left to right, would be done in one pass
    - Every path in a directed acyclic graph is a subsequence of topologically sorted vertex order, so processing vertexes in that order, we will do each path in forward order (will never relax edges out of vertex before doing all edges into vertex).
    - Thus: just one pass. *What will be the running time?*

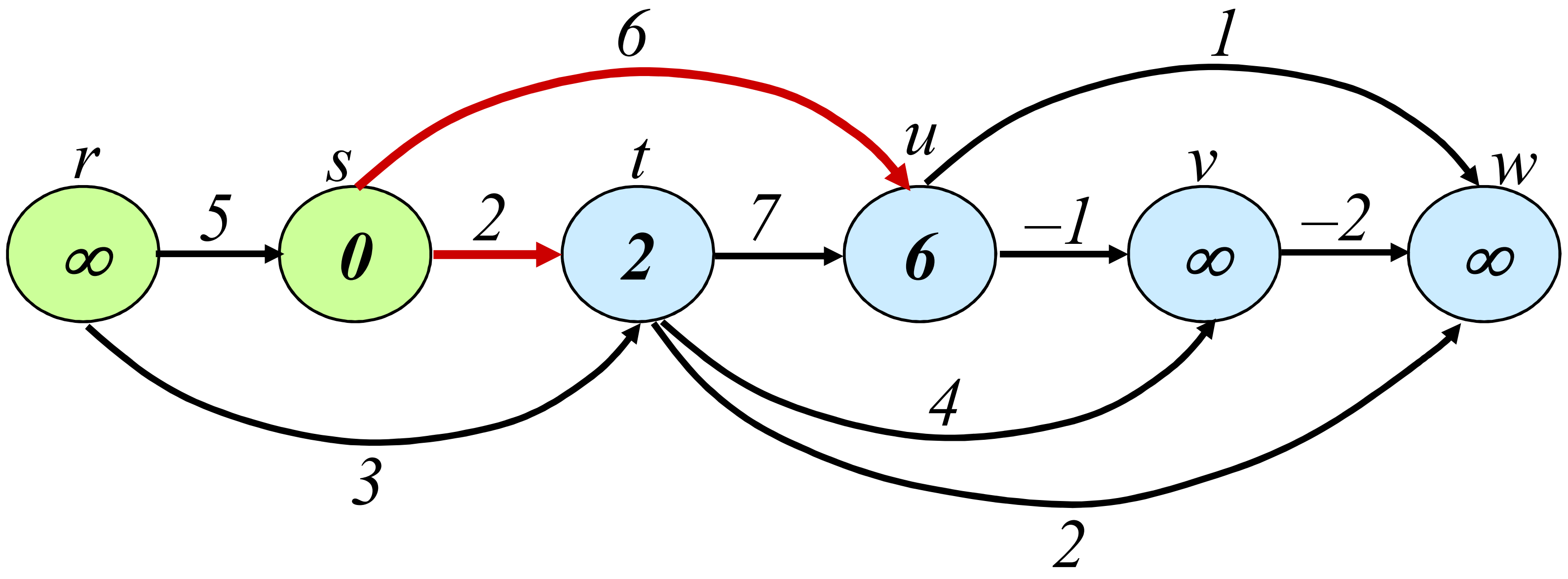
Example



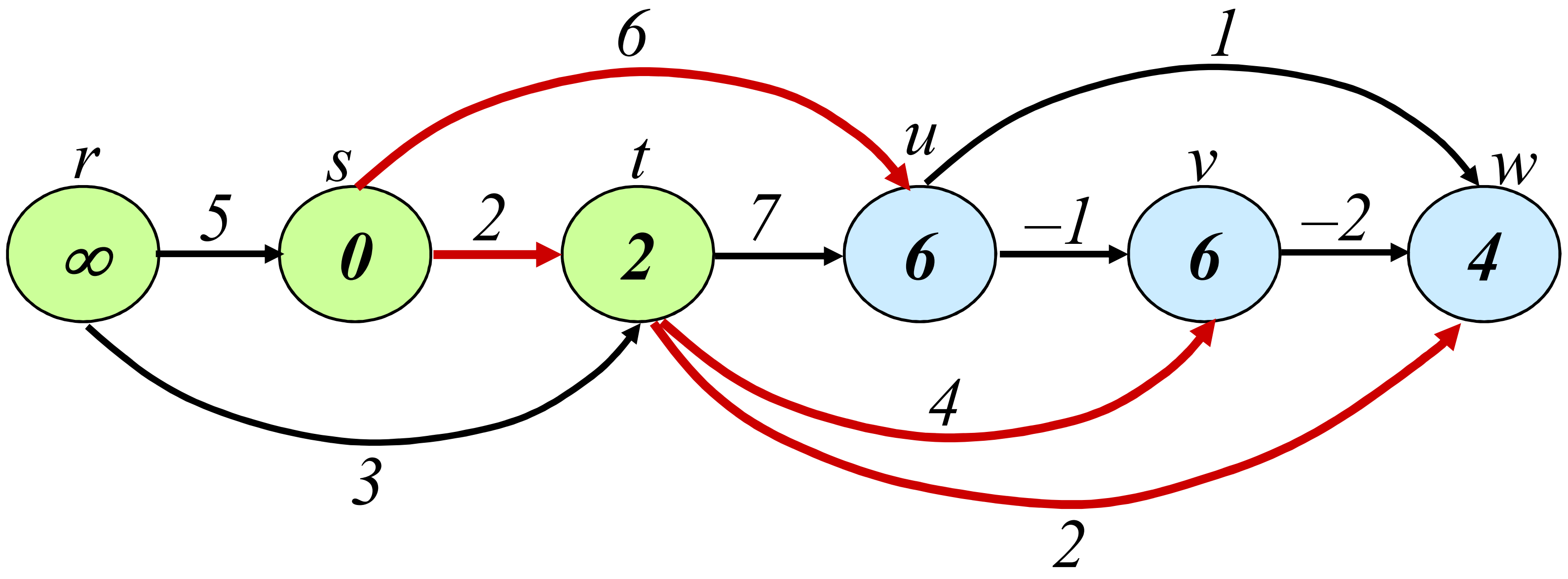
Example



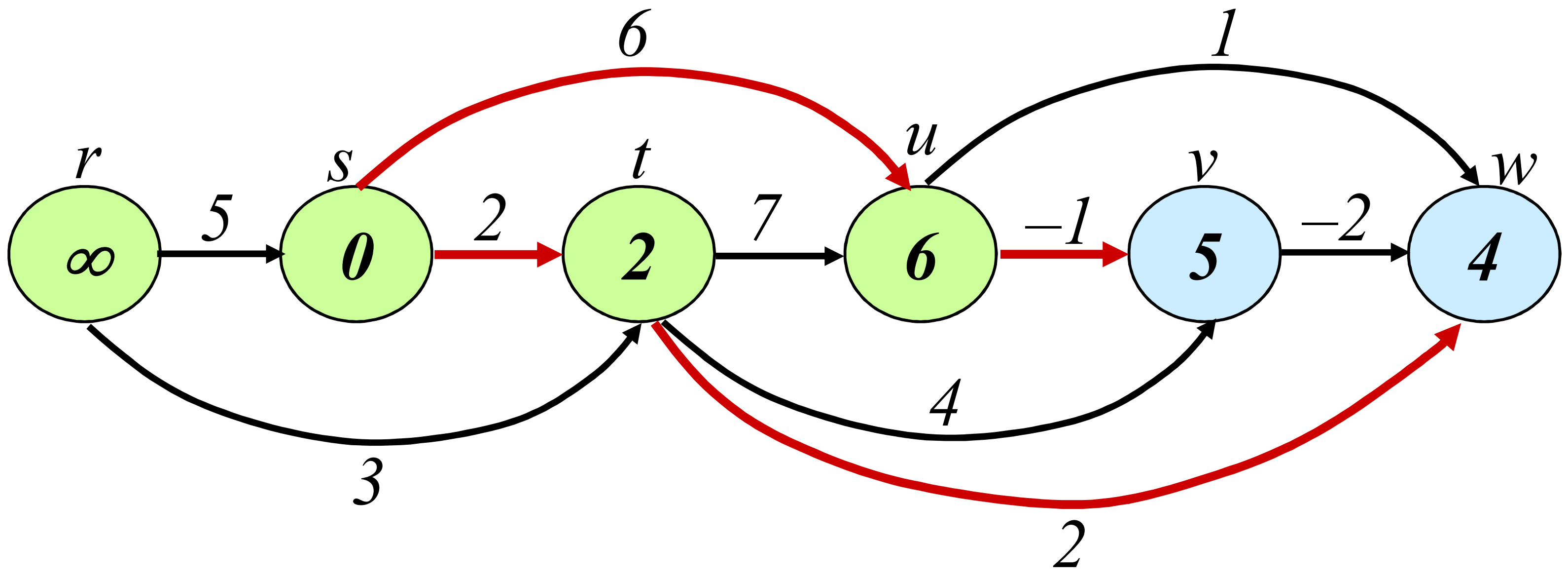
Example



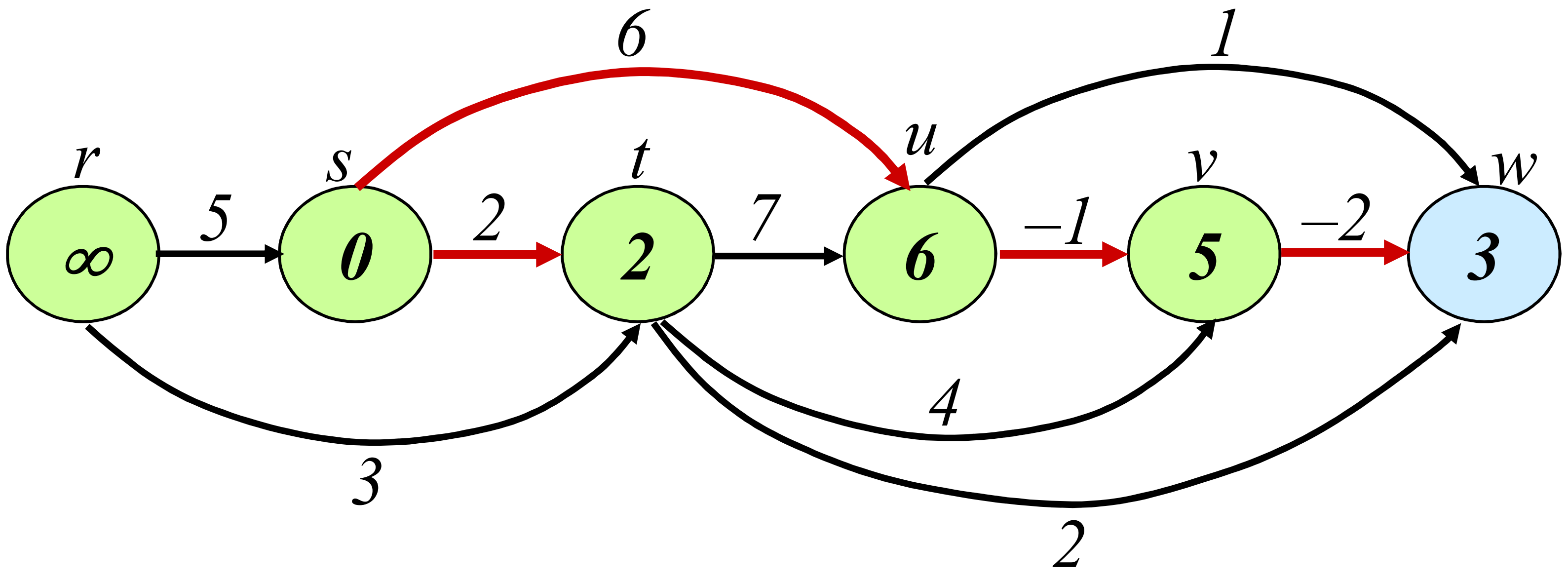
Example



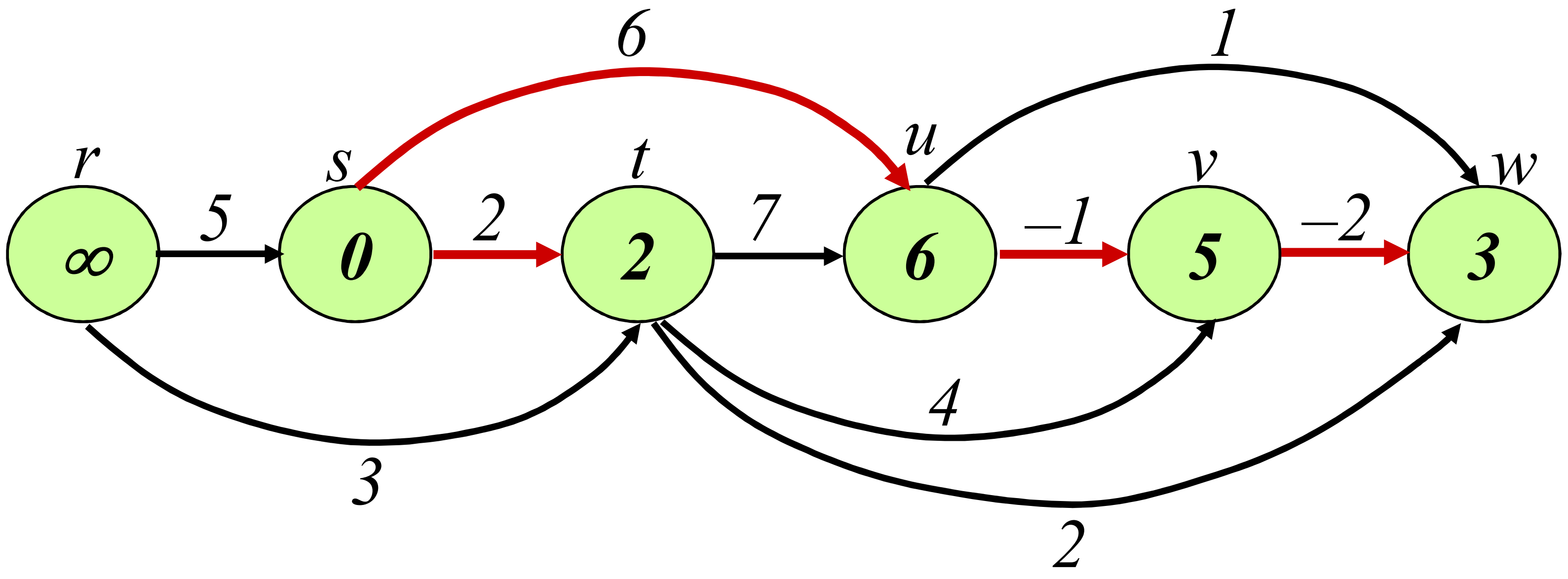
Example



Example



Example



DAG Shortest Paths

```
DAG-SHORTEST-PATHS(V,E,w,s)
    topologically sort the vertices
    INIT-SINGLE-SOURCE(V,s)
    for each vertex u,
        take in topologically sorted order
        do for each vertex v <- Adj[u]
            do RELAX(u,v,w)
```

Time complexity: O(V+E)

Source: https://zoomkoding.github.io/algorithm/2019/05/09/shortest-paths-1.html

Summary: Bellman-Ford

- Running time: O(VE)
  - Not so good for large dense graphs
  - But a very practical algorithm in many ways
- Note that order in which edges are processed affects how quickly it converges (show example)
  - Using topological sort: O(V+E)