# Ch17: Single Source Shortest Path

305234

Algorithm Analysis and Design

Jiraporn Pooksook
Naresuan University

# Shortest Path Problem

- A motorist wishes to find the shortest possible route from Chicago to Boston.
- Given a road map of the US on which the distance between each pair of adjacent intersections is marked.
- How can we determine the shortest route?

# Shortest Path Problem

- Given a weighted directed graph G =(V,E) with weight function

  w : E -> R

- Mapping edges to read valued weights.
- Let path p = ($v_0$ , $v_1$ ,…, $v_k$ ) and ($v_i$ , $v_{i+1}$ ) ϵ E for 0 ≤ i < k
- The weight of path p = ($v_0$ , $v_1$ ,…, $v_k$ ) is the sum of the weights of its constituent edges:

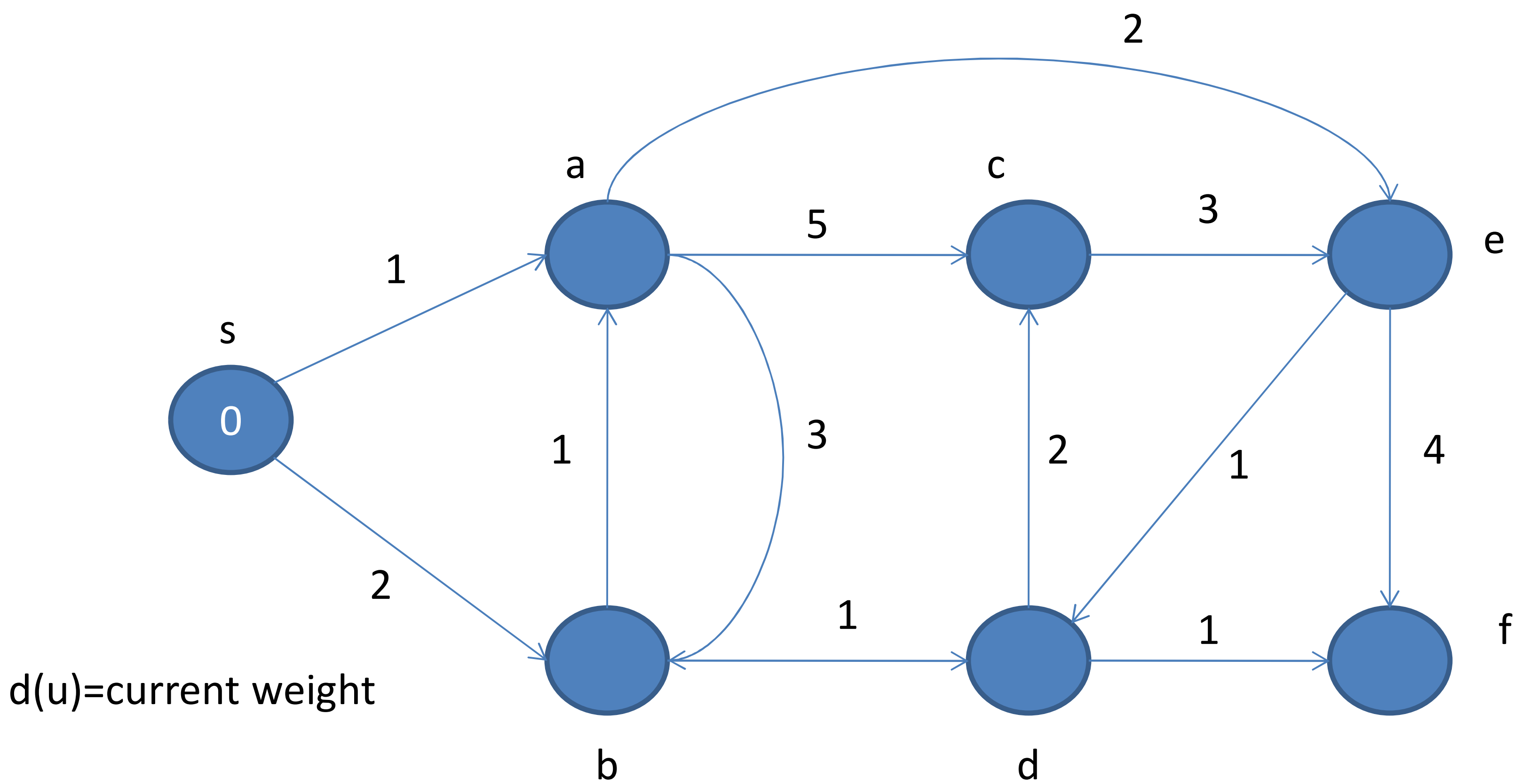
$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

- We define the shortest-path weight from u to v by

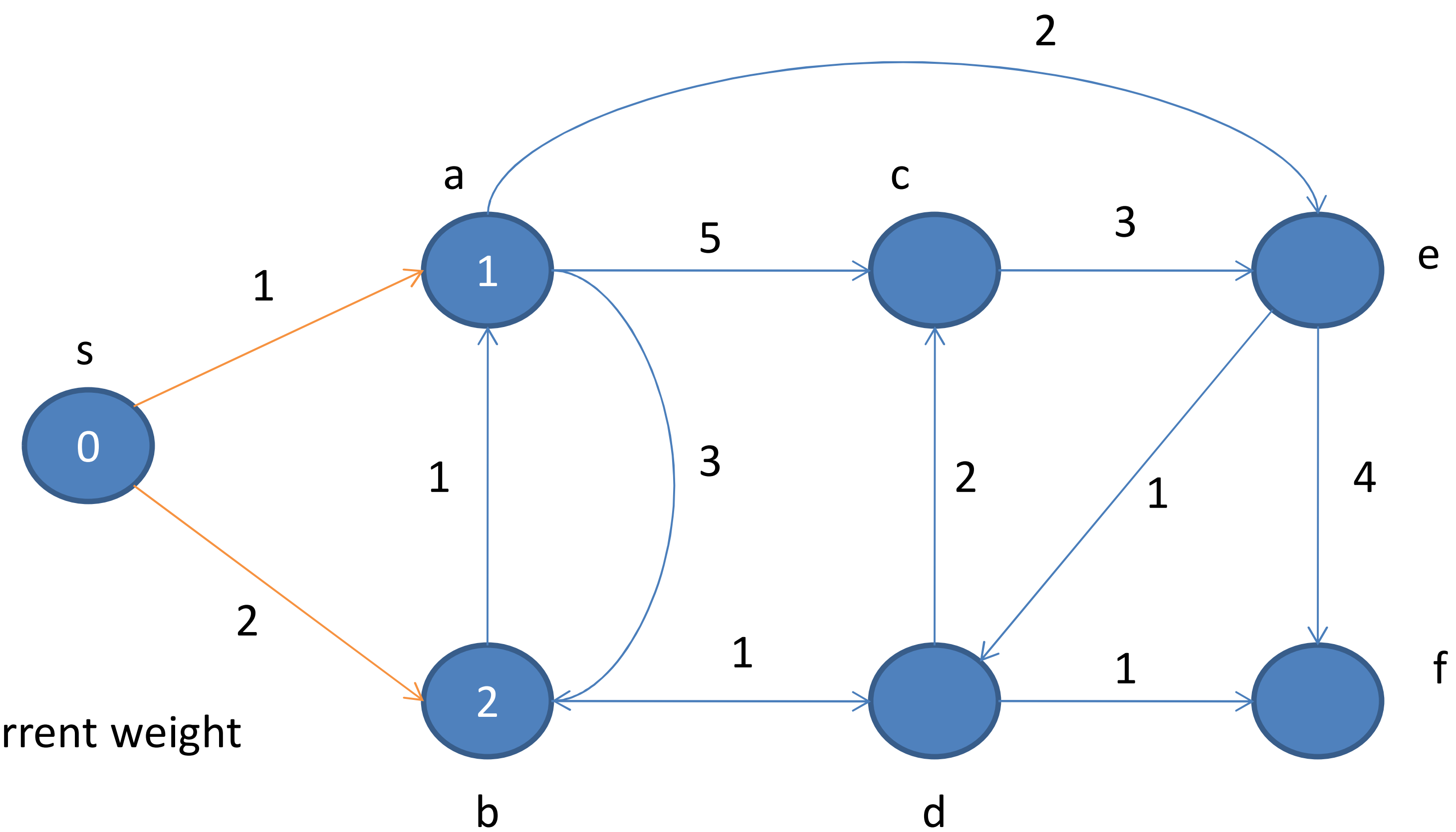
$$\delta(u,v) = \min\{w(p) : u \xrightarrow{p} v\}$$

If there is a path from u to v , otherwise $\delta(u,v) = \infty$

# Shortest Path Problem

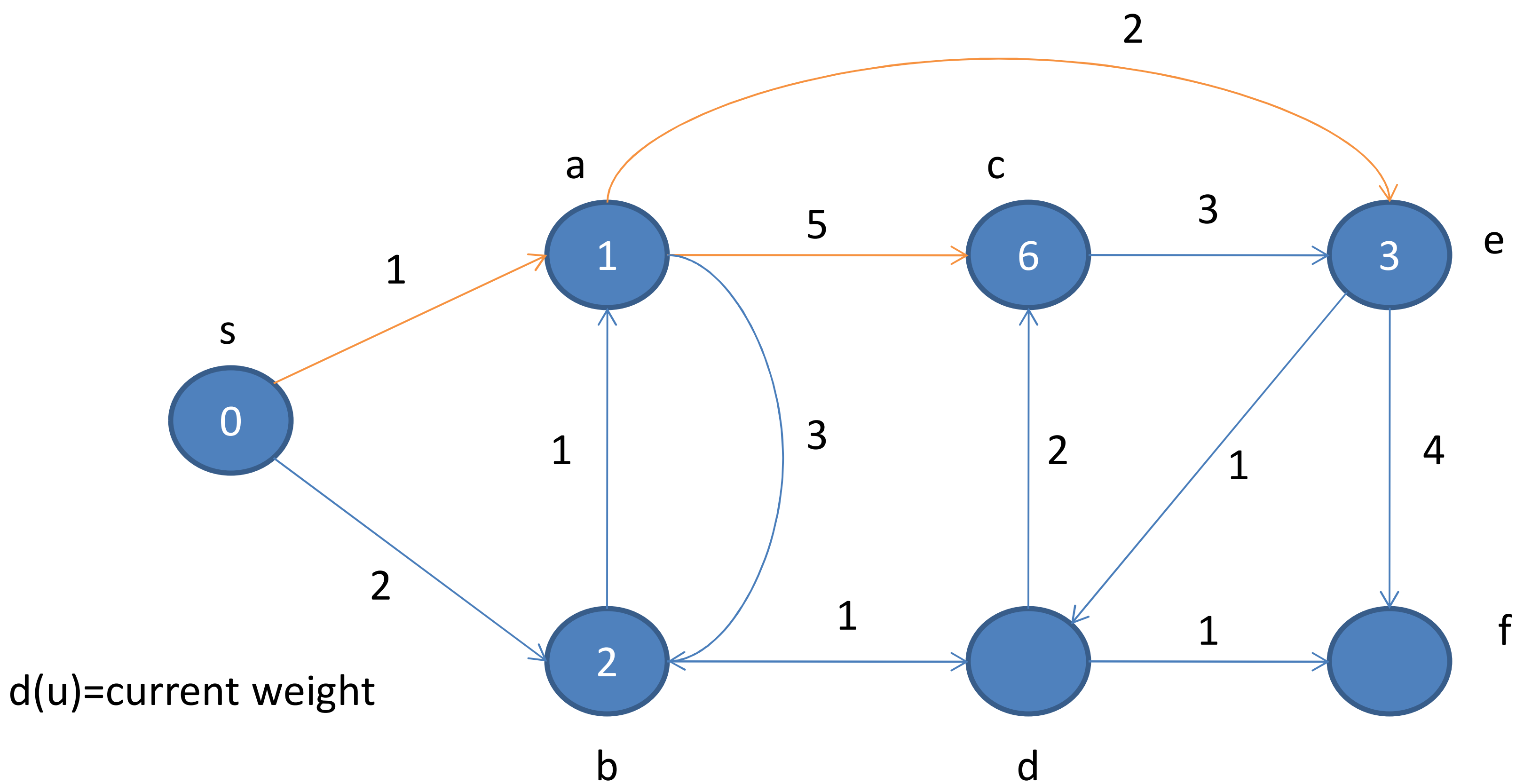


# Shortest Path Problem



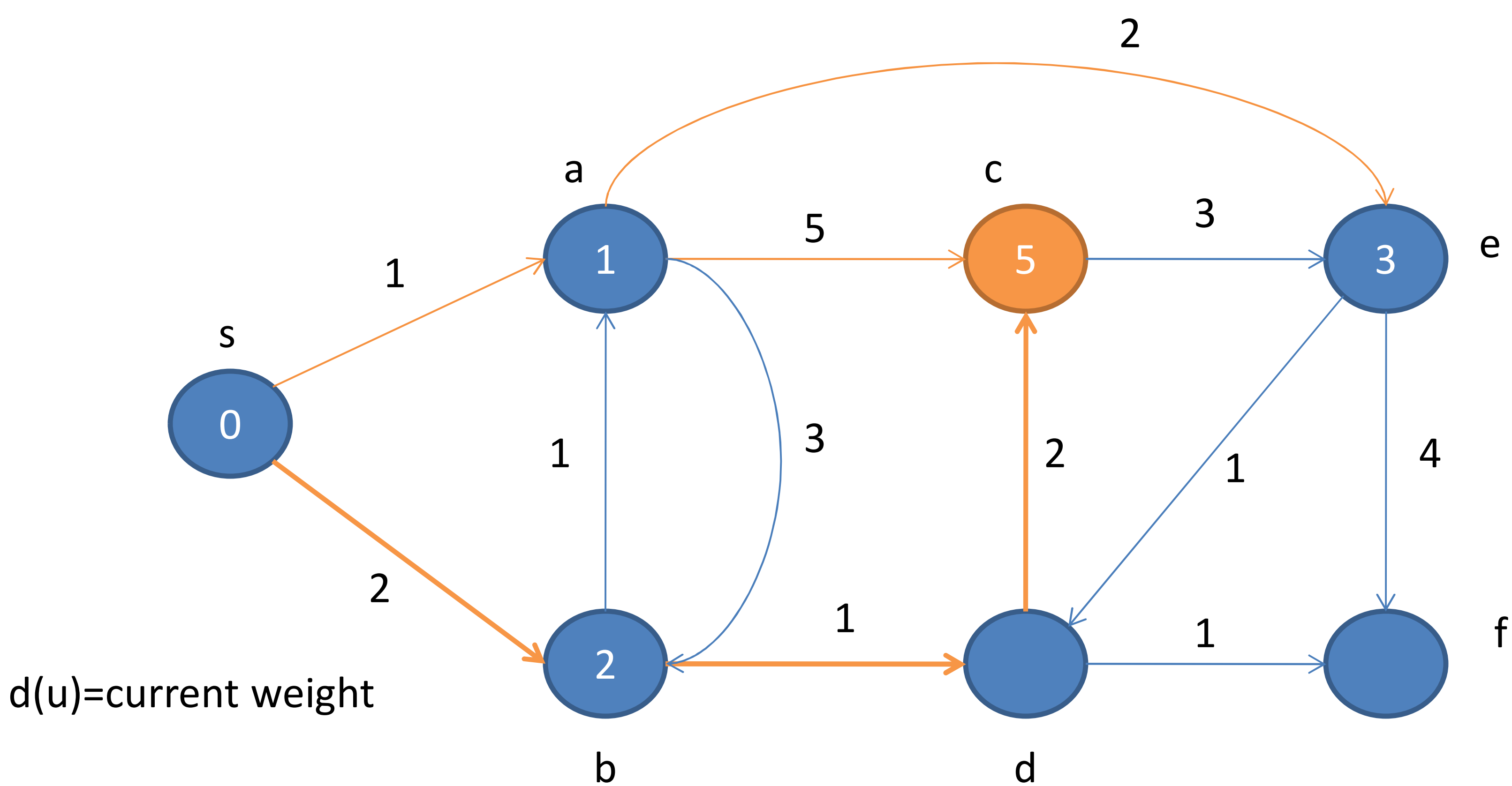

d(u)=current weight

# Shortest Path Problem




$$\delta(s,c)=6?$$ Can we find a shorter path?

# Shortest Path Problem

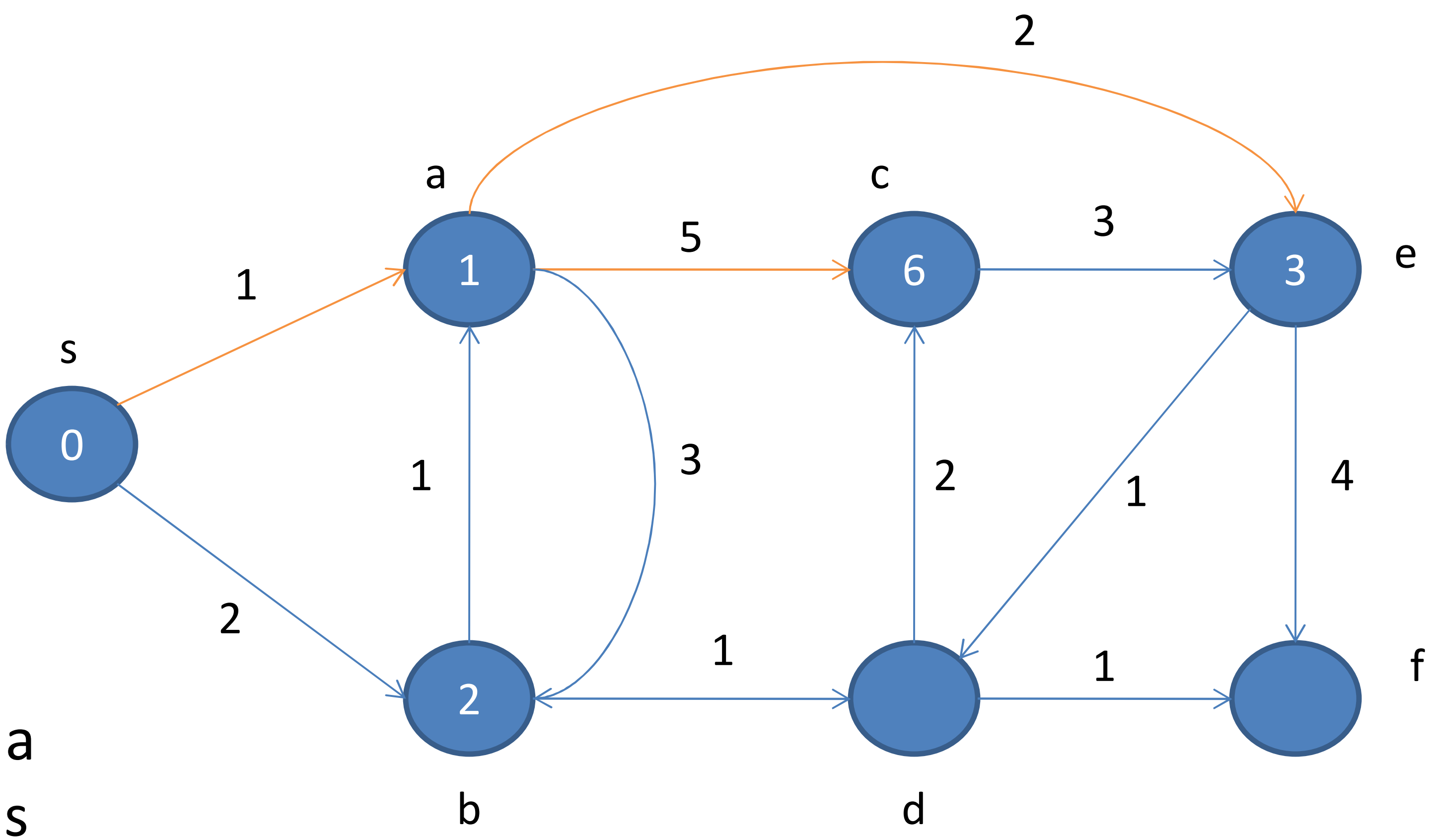



$$\boxed{\delta(s,c) = 5}$$

# Representing Shortest Path

- Given a graph G=(V,E)
- For each vertex v ϵ V, a **predecessor** ¶[v] that is either another vertex or NIL.
- We denote d(v) as a value inside a circle(graph) to be a current weight.
- We denote ¶[v], for any vertex v, as a predecessor on the current best path to v.
- ¶[s]=NIL

# Representing Short Path

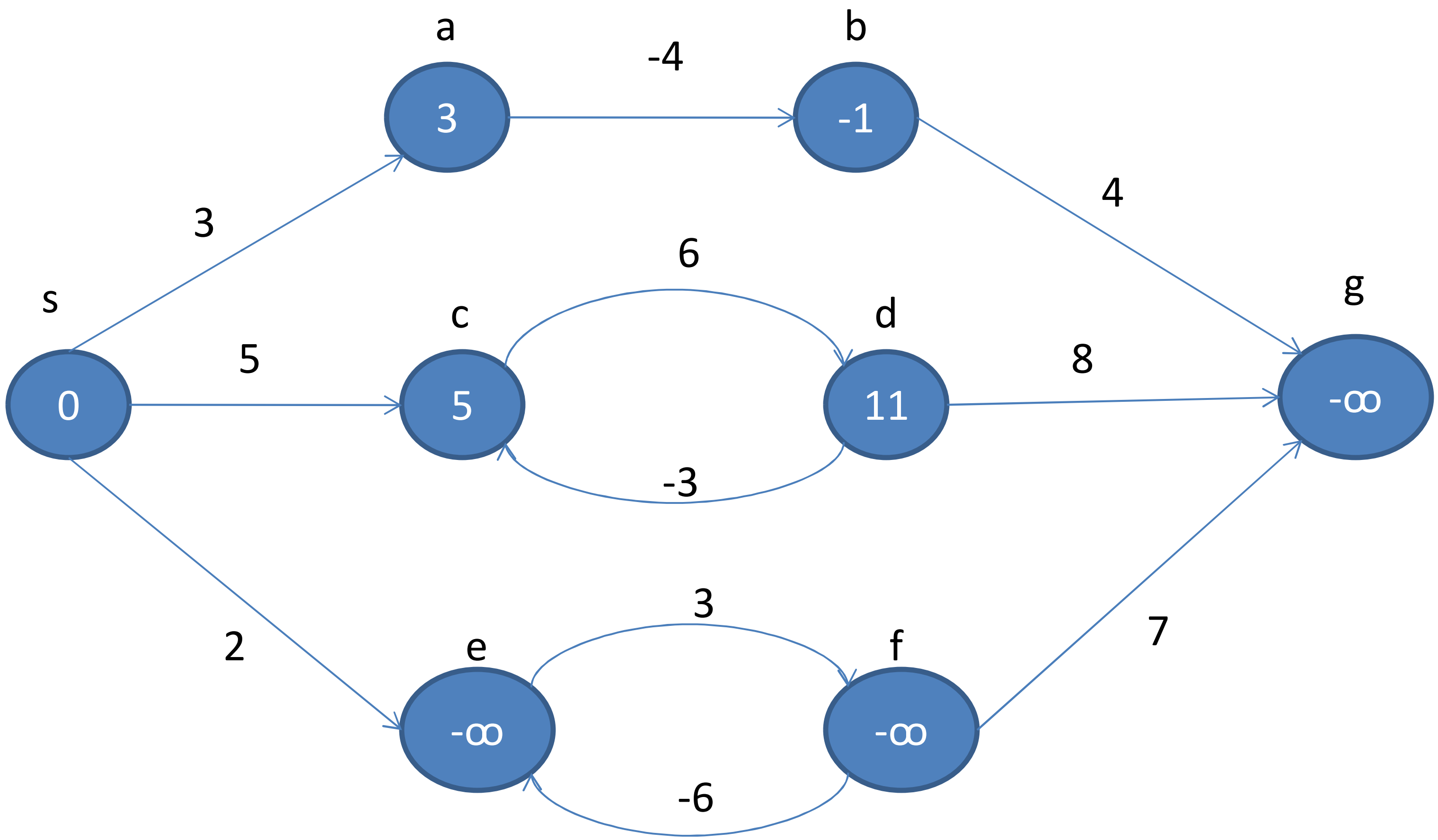



¶[e] = a
¶[a] = s

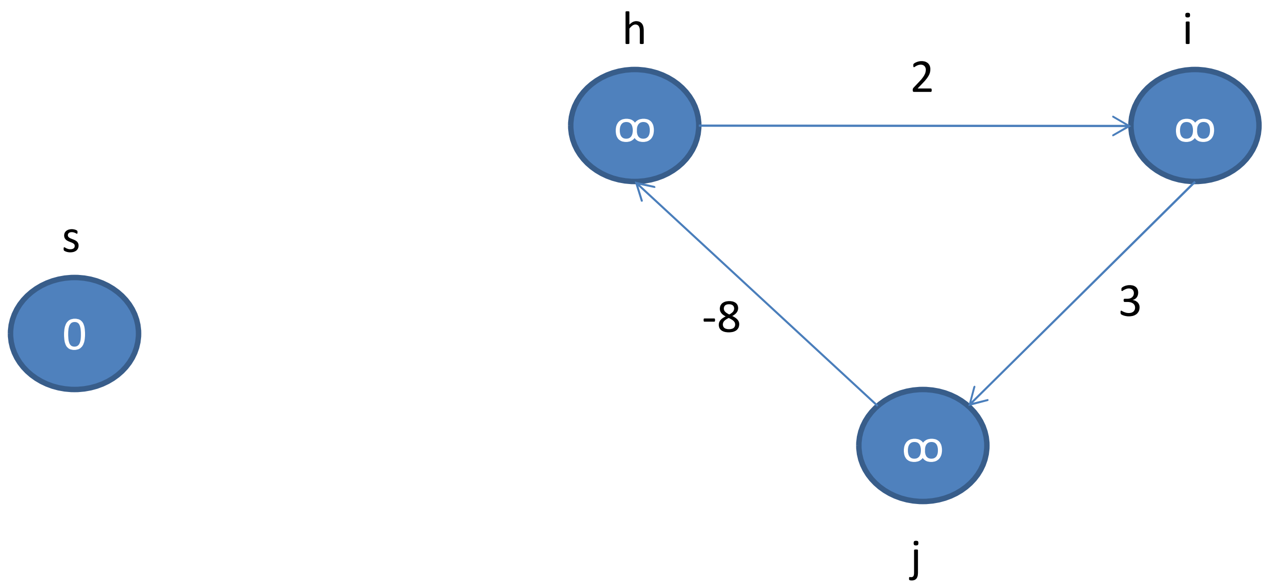
# Negative-weight Edges

- There may be edges whose weights are negative.
- If there is a negative-weight cycle reachable from s, shortest-path weights are not well defined.
- If there is a **negative-weight cycle** on some path from s to v, we define $\delta(s,v) = -\infty$

# Negative-weight Edges



# Negative-weight Edges



# General Structure of Shortest Path

- Initialize single source
  - For u ϵ V, we set d[v] = ꝏ, ¶[u] = NIL and d[s] = 0
- Relaxation
  - Repeatedly select edge(u,v) and relax(u,v) by checking the condition:

    if d[v] > d[u] + w(u,v)

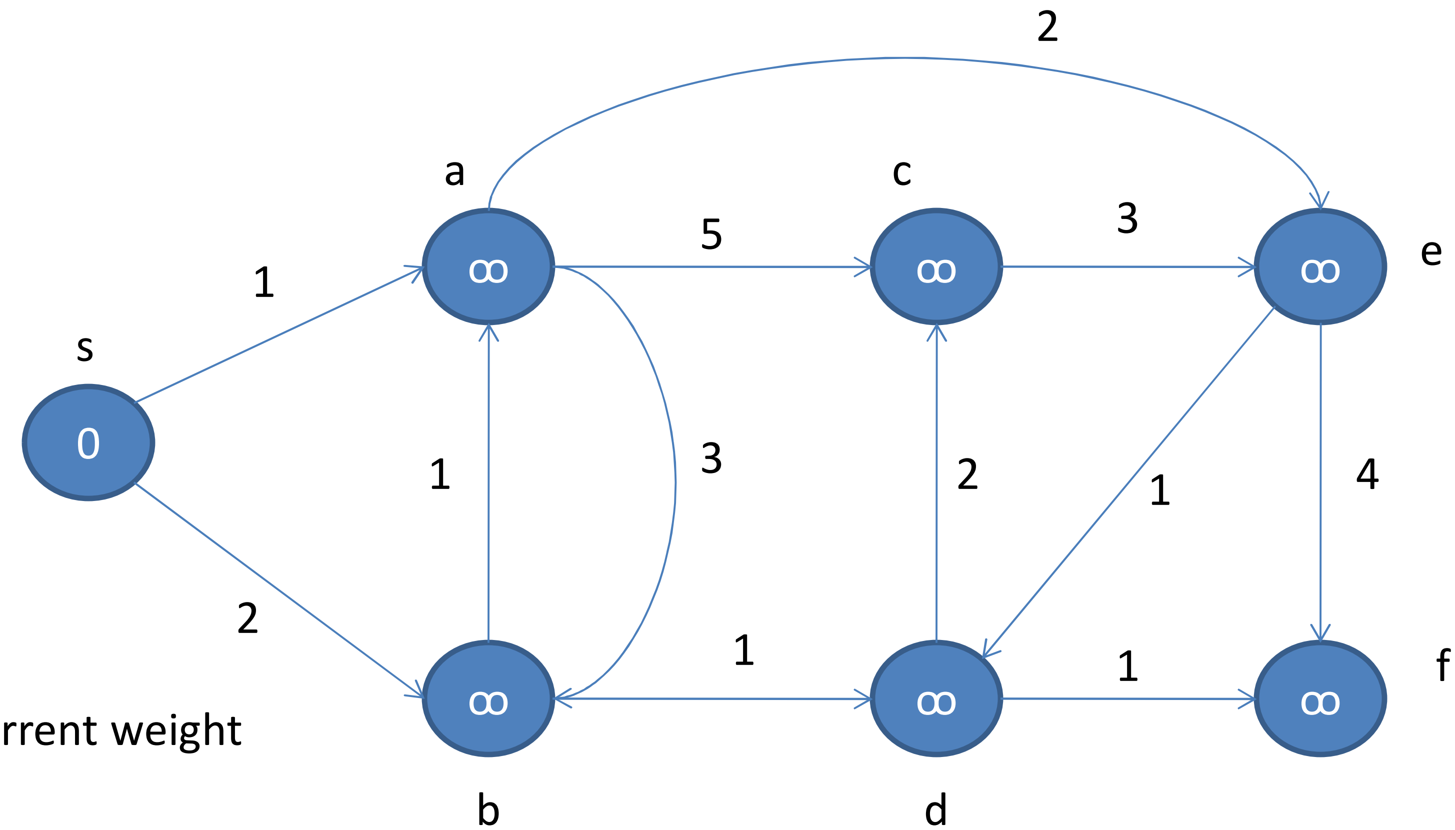
    then d[v] = d[u] + w(u,v)

    ¶[v] = u

# Initialize-Single-Source(G,s)

```
for each vertex v in V[G]
        do d[v] = ∞
           ¶[v] = NIL
d[s] = 0
```

# Initialize-Single-Source(G,s)

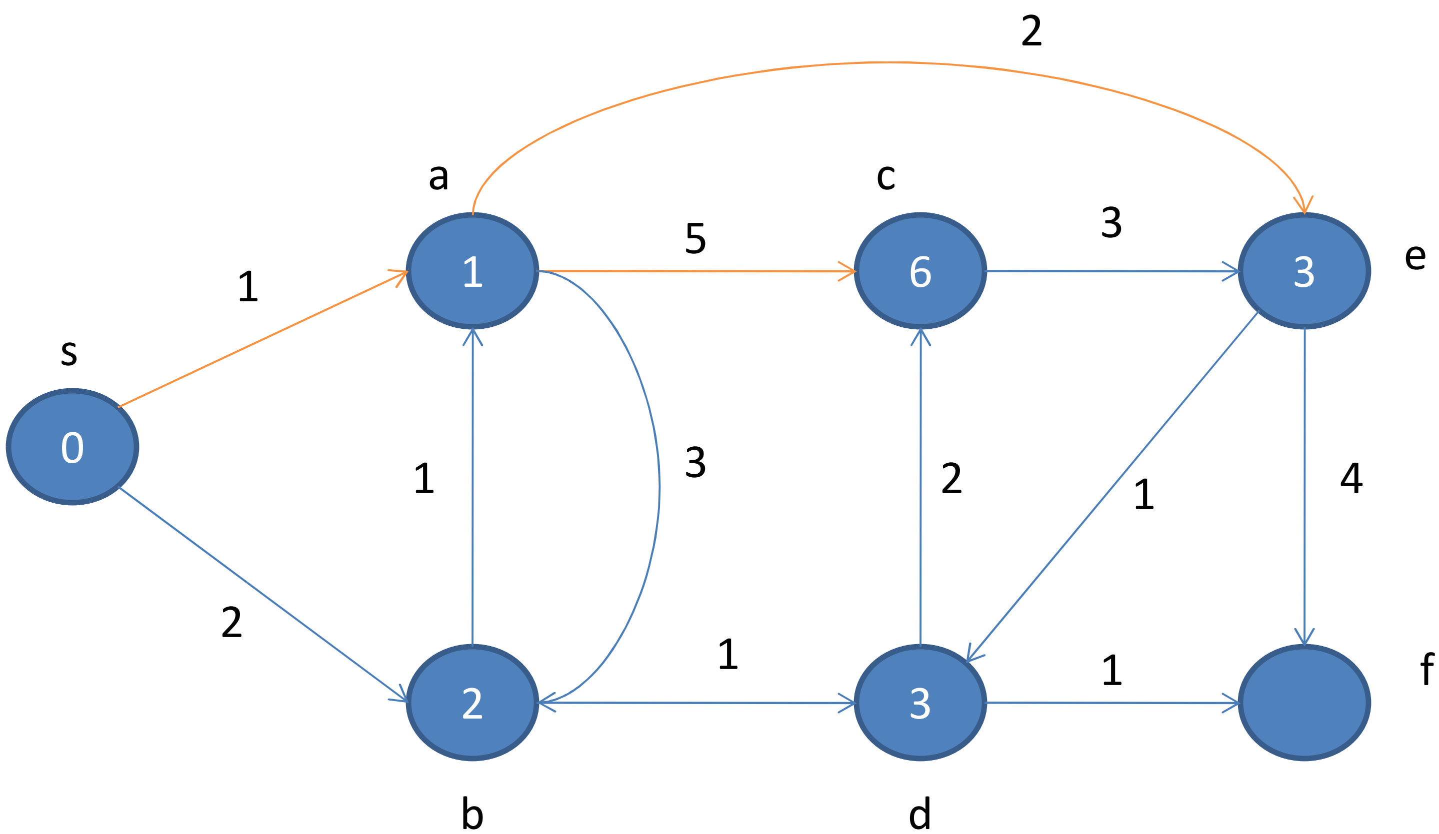



d(u)=current weight

# Relaxation(u,v,w)

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

# Relaxation(d, c, w(d,c) )




d[c] > d[d] + w(d,c)
6 > 3 + 2
Hence, d[c] = 5 and ¶[c] = d

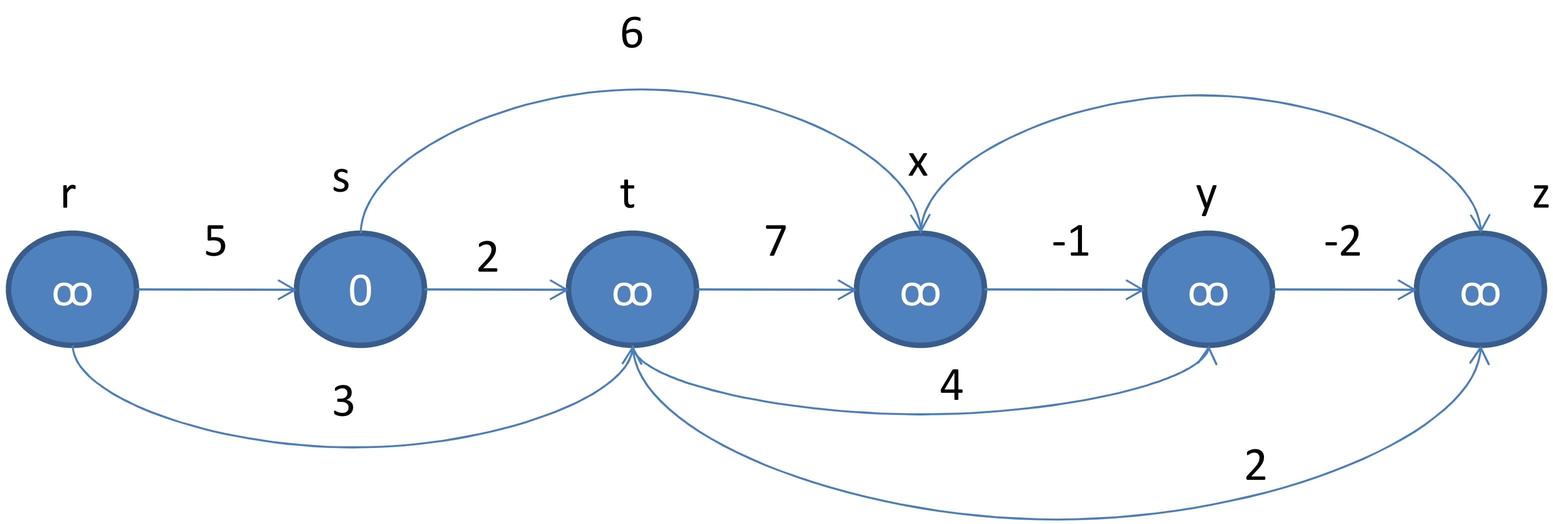
# Shortest path in Directed Acyclic Graphs

- We can compute shortest paths from a single source in O(V+E) time using relaxation on edges of a weighted **directed acyclic graph**(dag).

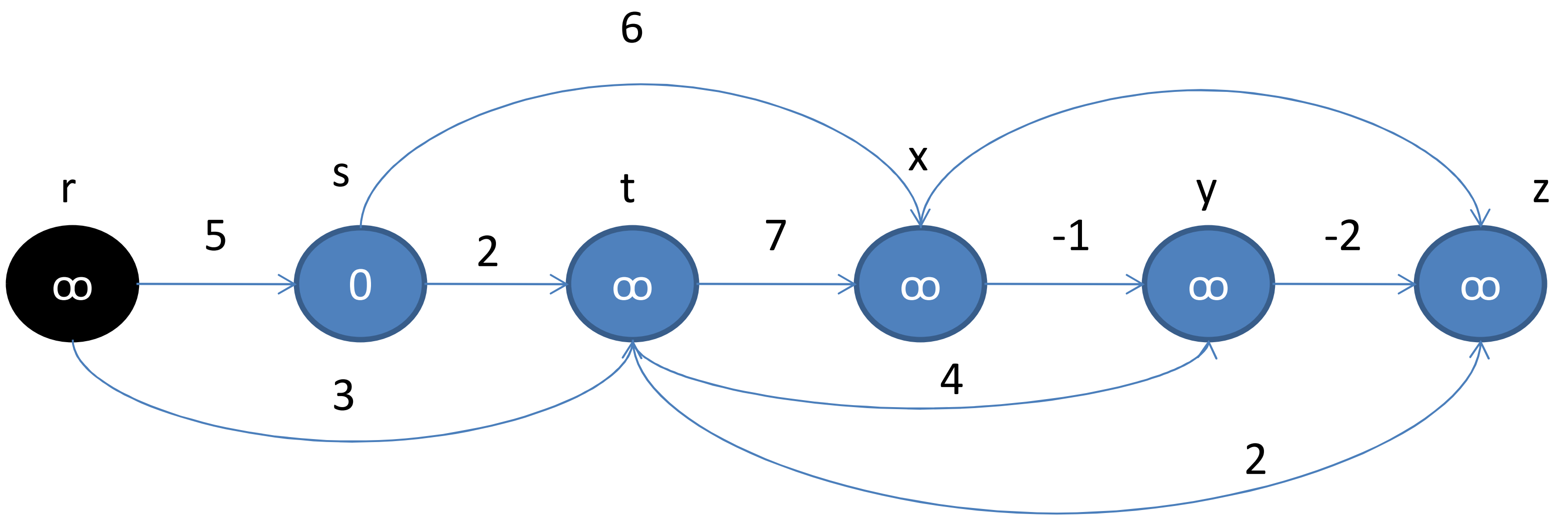
# DAG-SHORTEST-PATHS(G,w,s)

```
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G,s)
for  each vertex u, taken in topologically sorted order
        do for each vertex v ϵ Adj[u]
                do RELAX(u, v, w)
```

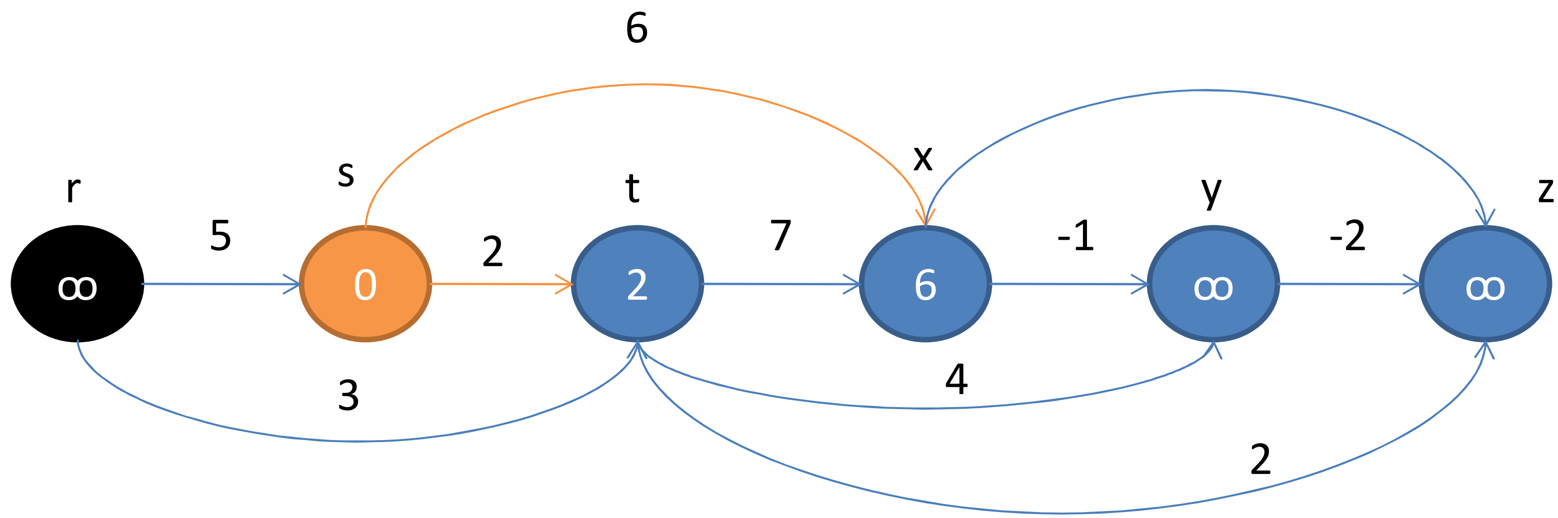
# Example: DAG



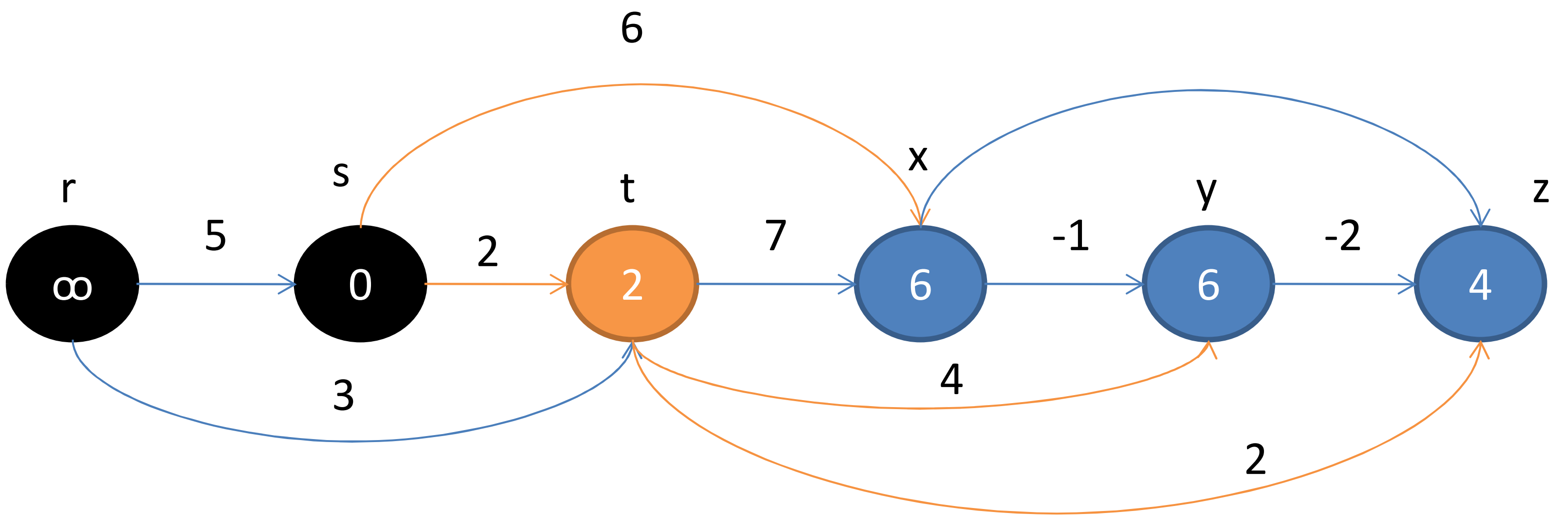
# Example: DAG



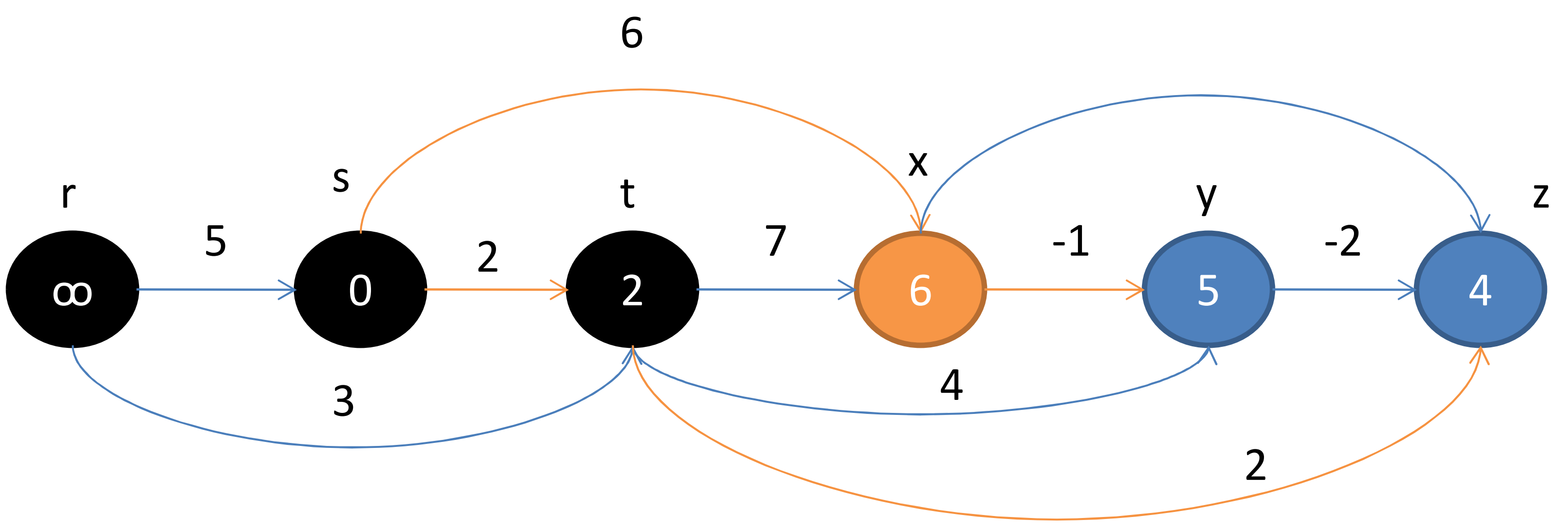
# Example: DAG



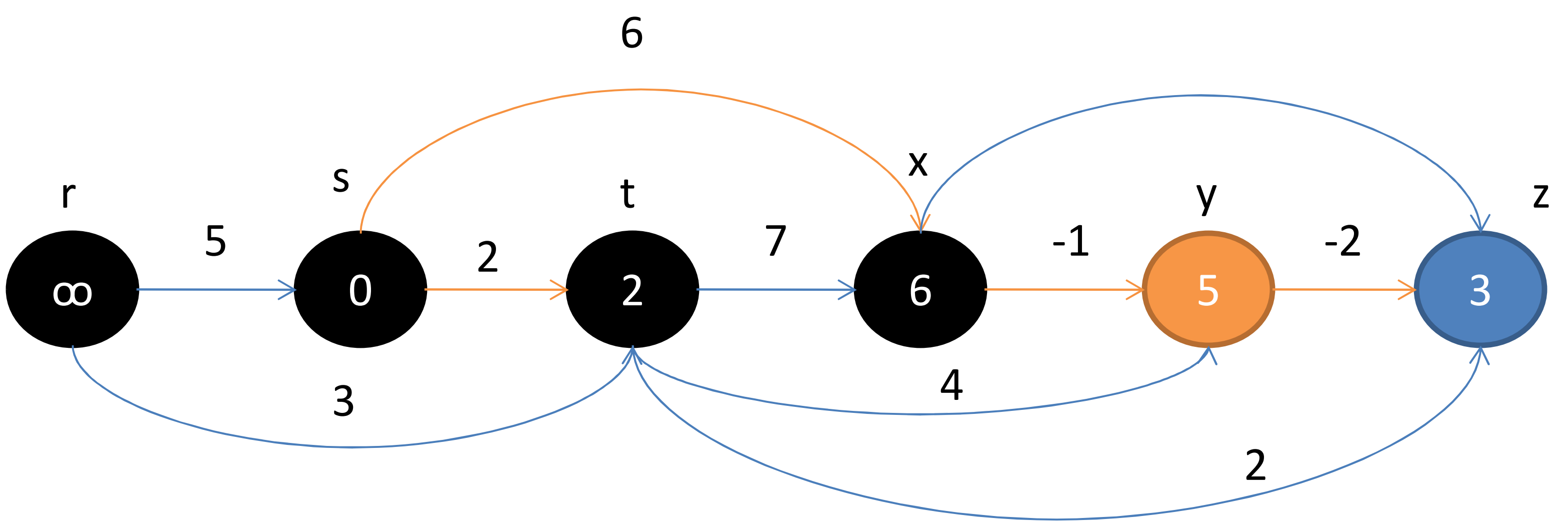
# Example: DAG



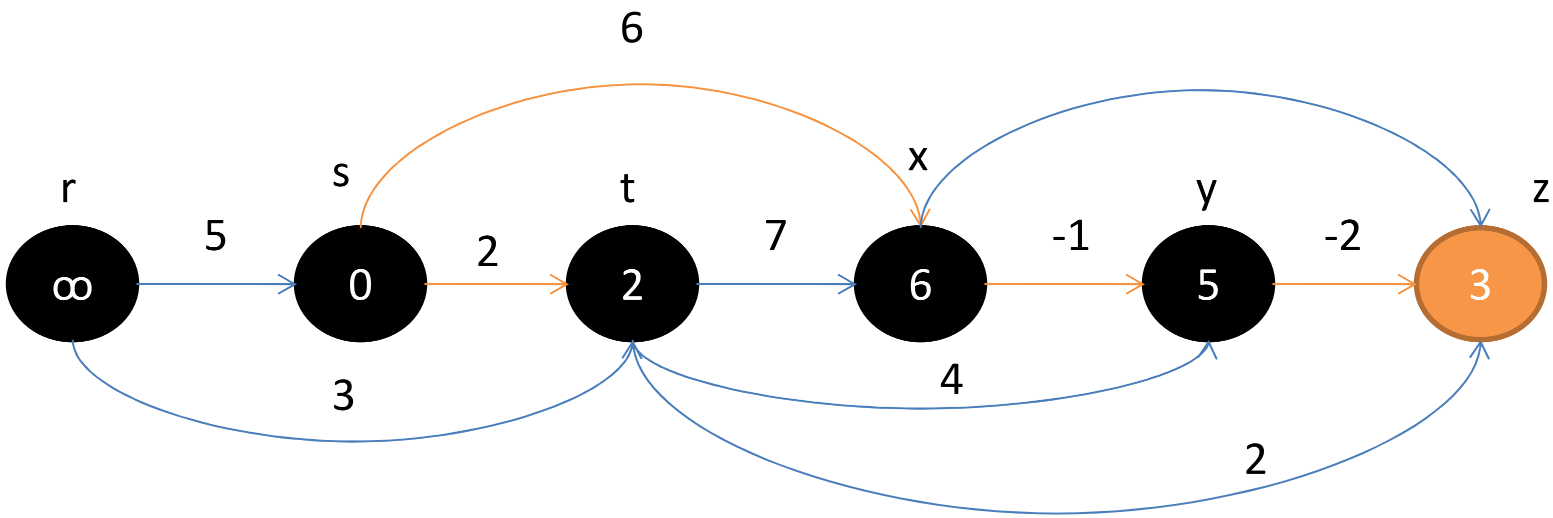
# Example: DAG



# Example: DAG



# Example: DAG



# Dijkstra Algorithm

- Solves the single-source shortest-paths problem on a weighted directed graph G = (V,E) for the case in which **all edge weights are nonnegative**.
- We assume that $w(u,v) \geq 0$ for each edge $(u,v) \in E$.

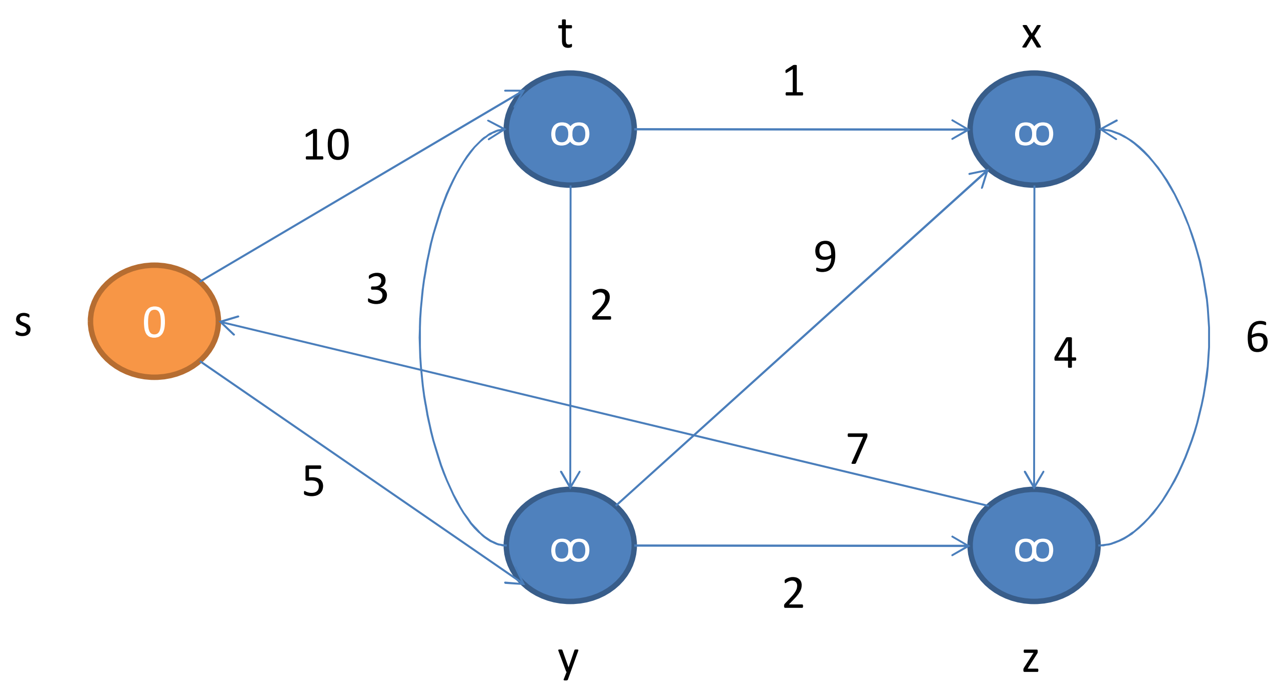
# Dijkstra(G,w,s)

```
INITIALIZE-SINGLE-SOURCE(G,s)
S = ∅
Q = V[G]
while Q != ∅
        do u = EXTRACT-MIN(Q)
            S = S U {u}
            for each vertex v ϵ Adj[u]
                do RELAX(u,v,w)
```

# Analyze Dijkstra

- The running time of Dijkstra depends on how to implement the min-priority queue.
- If we implement the min-priority queue with a binary min-heap which has running time O(lg V) if all vertices are reachable from the source. Hence total time is O((V+E)lg V) = O( E lg V)

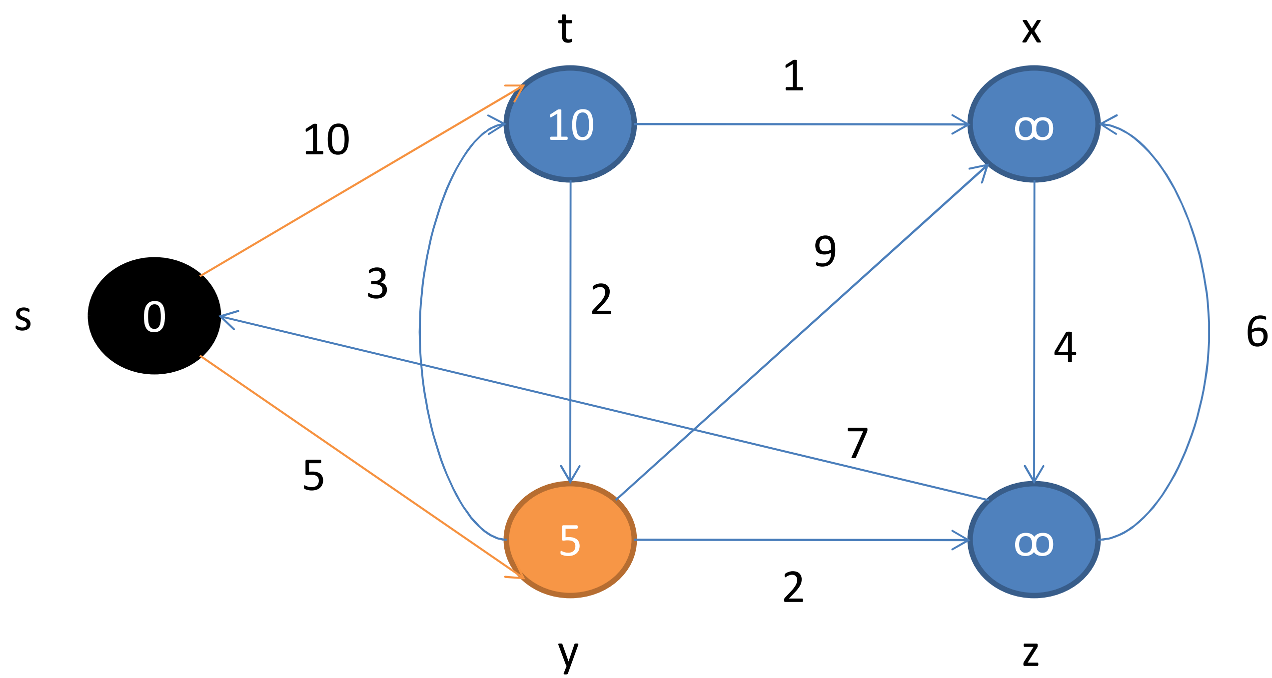
# Example: Dijkstra




S ={ }

Q = {0, ∞, ∞, ∞, ∞}

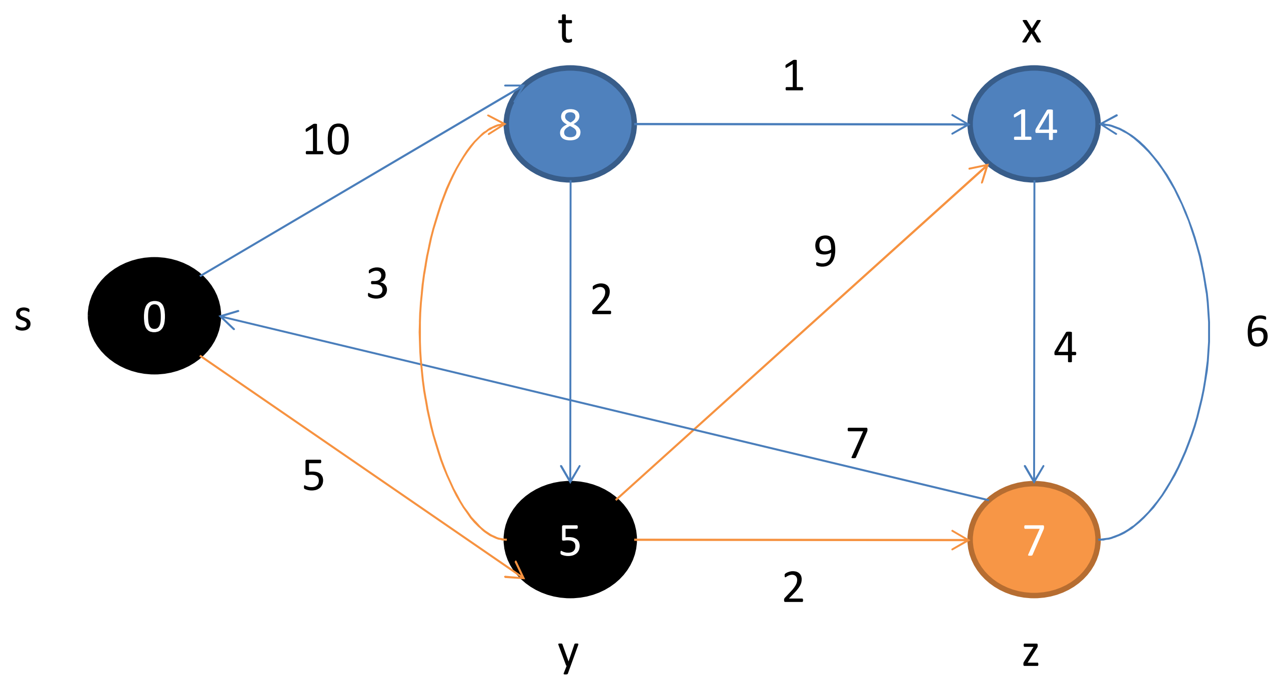
# Example: Dijkstra




S ={ s }

Q = {0, 10, 5, ∞, ∞}

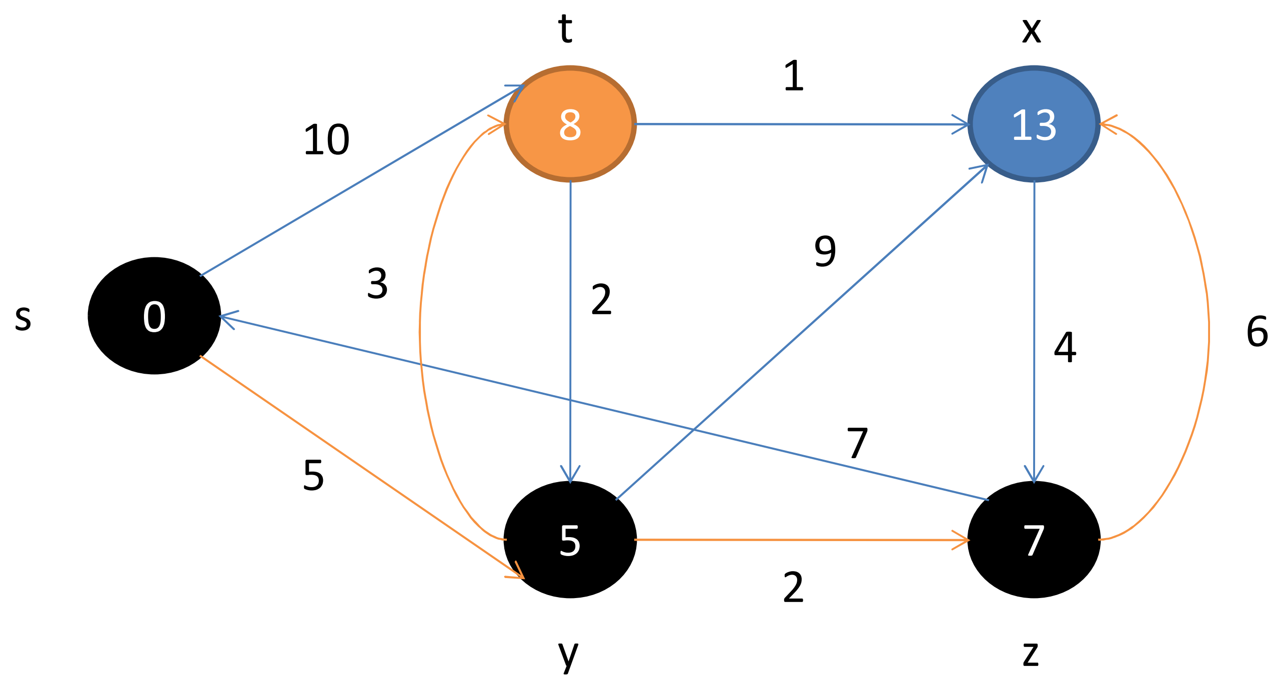
# Example: Dijkstra




S ={ s , y }

Q = {0, 8, 5, 14, 7}

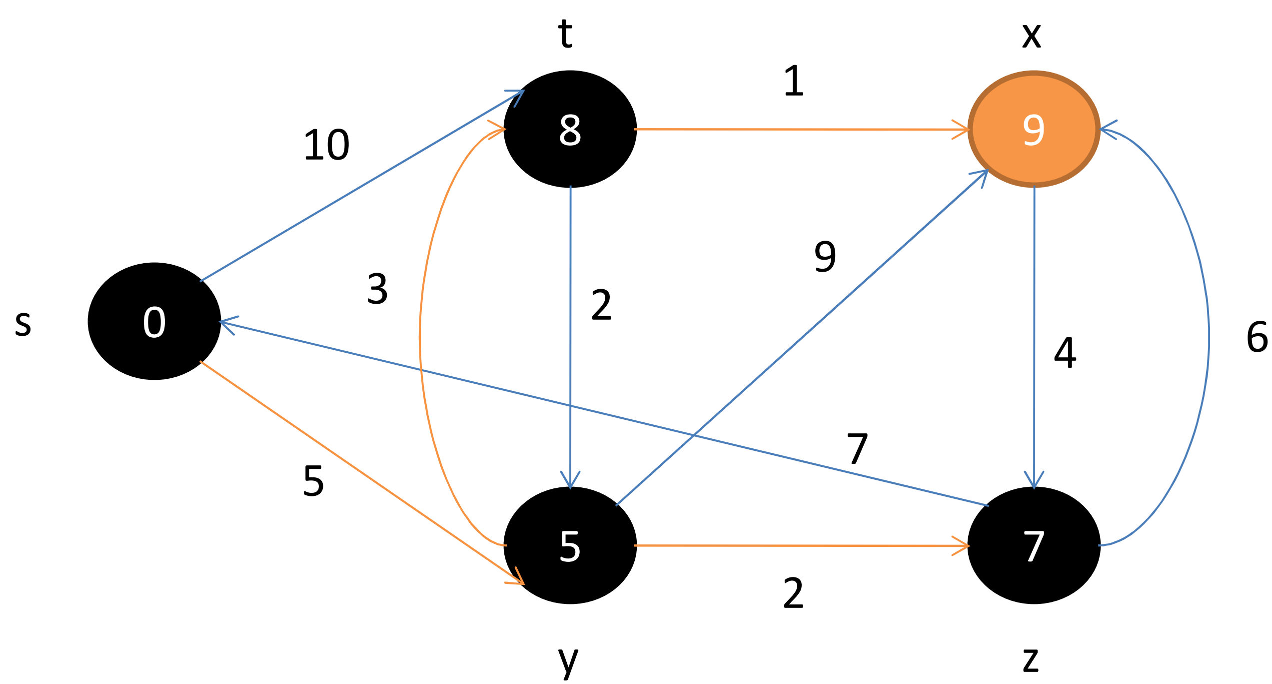
# Example: Dijkstra




S ={ s , y, z }

Q = {0, 8, 5, 13, 7}

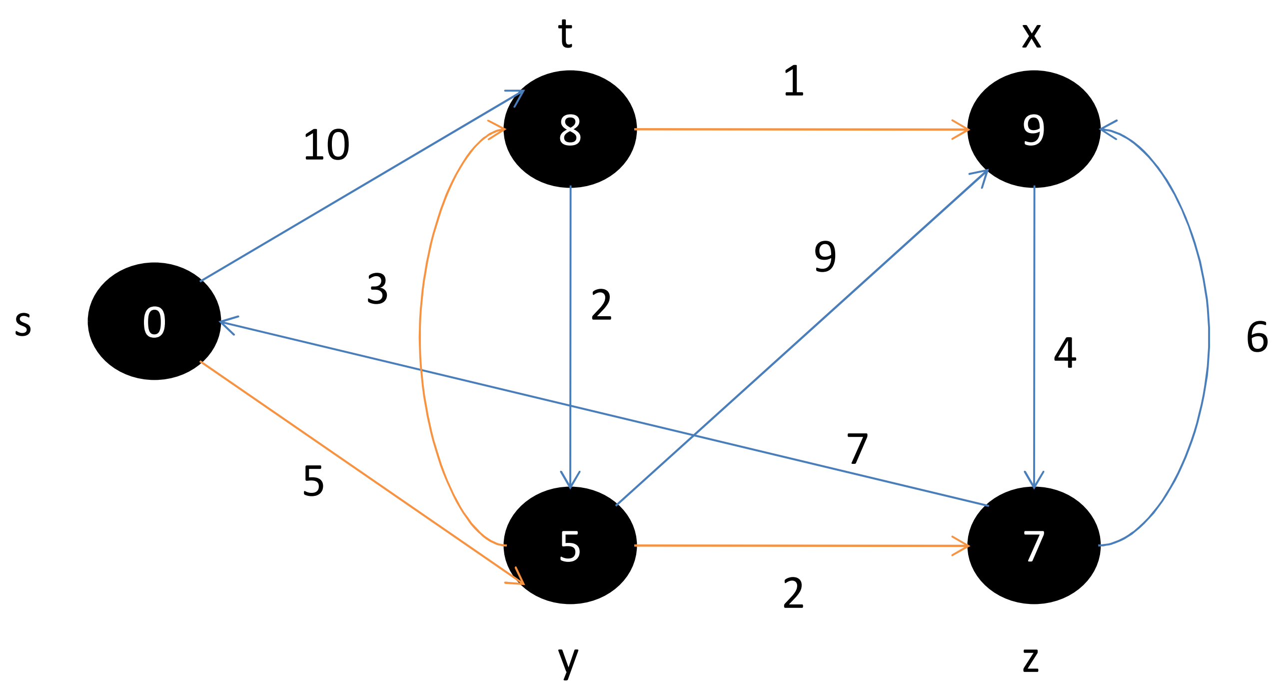
# Example: Dijkstra




S ={ s ,t, y, z }

Q = {0, 8, 5, 9, 7}

# Example: Dijkstra




S ={ s ,t,x, y, z }

Q = {0, 8, 5, 9, 7}

# Bellman-Ford Algorithm

- Solves the single-source shortest-paths problem in general case in which **edge weights may be negative**.
- The Bellman-Ford algorithm returns a boolean value indicating whether or not there is a negative-weight cycle that is reachable from the source. If there is such a cycle, the algorithm indicates that no solution exists. If there is no such cycle, the algorithm produces the shortes paths and their weights.

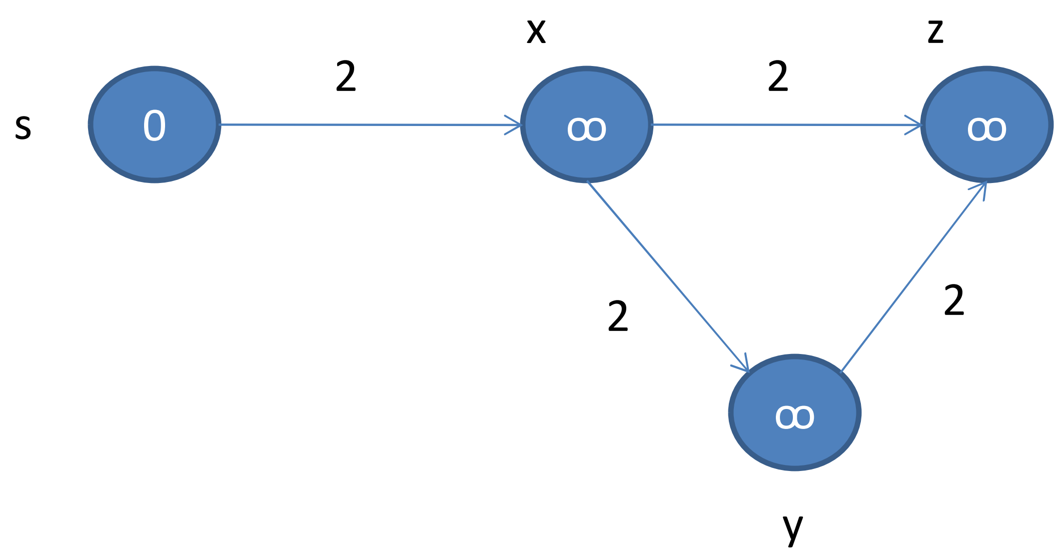
# Bellman-Ford(G,w,s)

```
INITIALIZE-SINGLE-SOURCE(G,s)
for i =1 to |V[G]| -1
          do  for each edge (u,v) ϵ E[G]
                    do RELAX(u, v , w)
for each edge (u,v) ϵ E[G]
          do if d[v] > d[u] + w(u,v)
                    then return false
return true
```

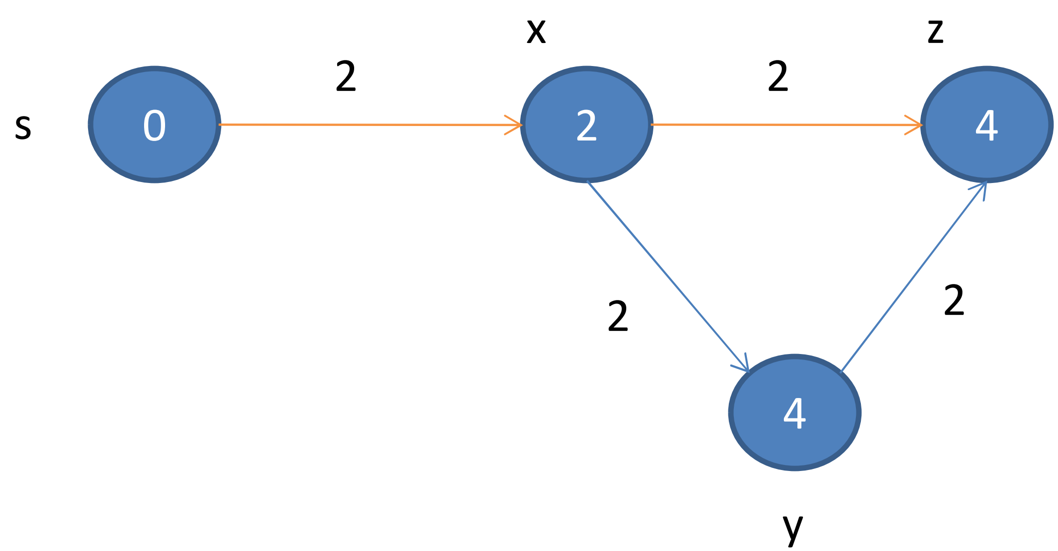
# Analyze Bellman-Ford

- The running time is O(VE) since the initialization take O(V) and each of |V|-1 passes over edges in lines 2-4 takes O(E), and for loop in lines 5-7 takes O(E) time.

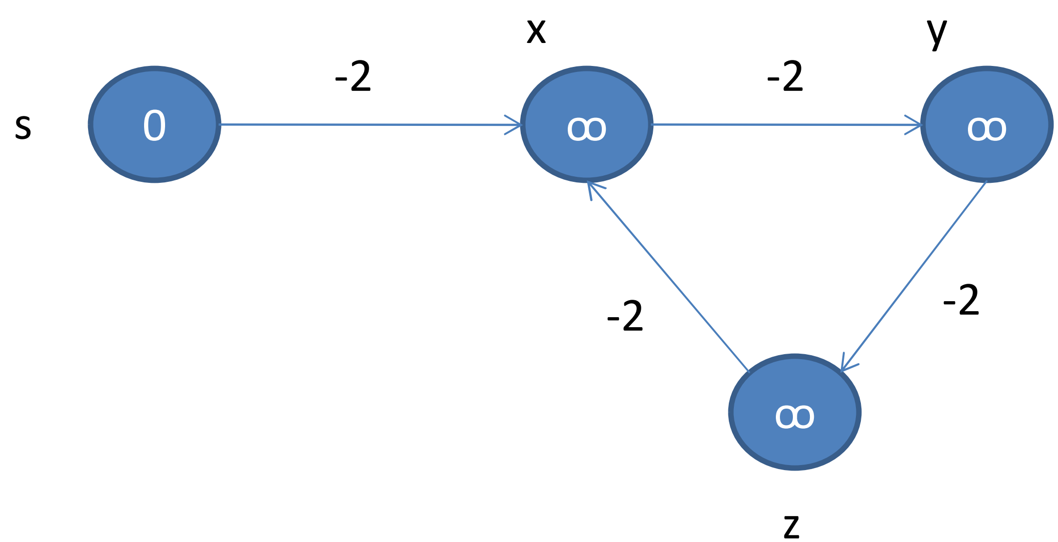
# Example: Bellman-Ford



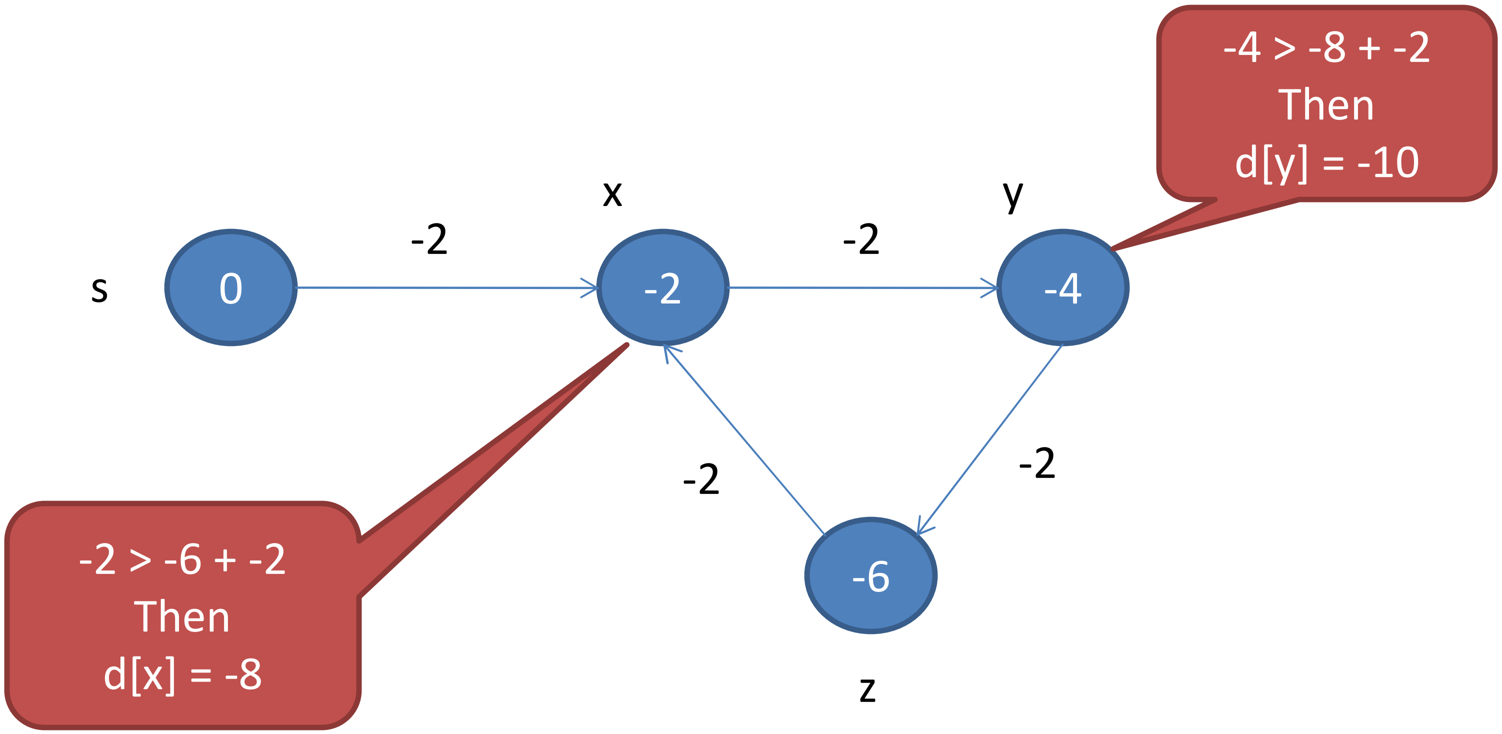
# Example: Bellman-Ford



# Example: Bellman-Ford

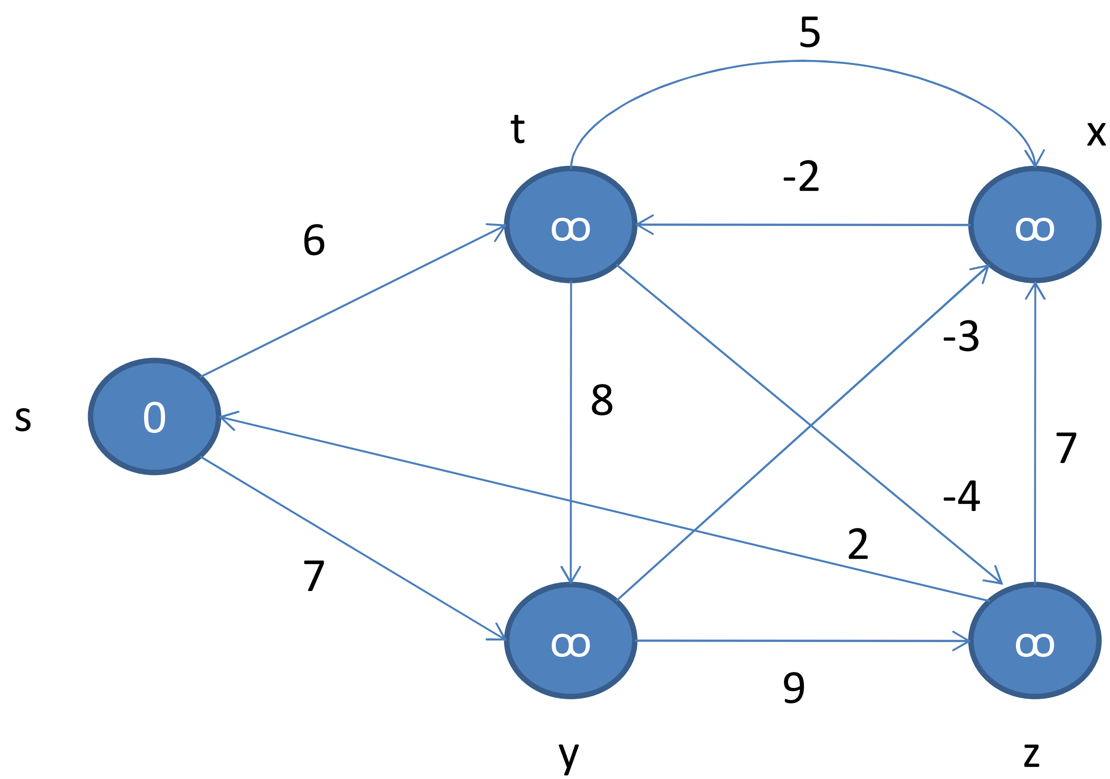


# Example: Bellman-Ford

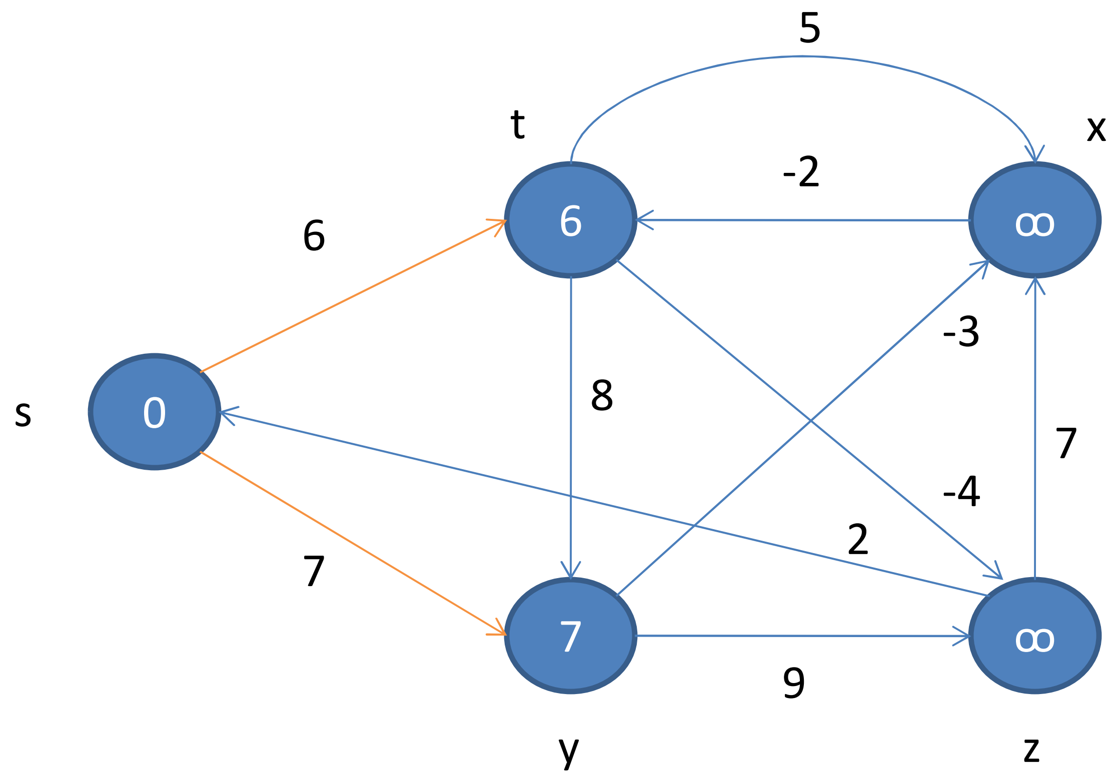



RETURN false !!!

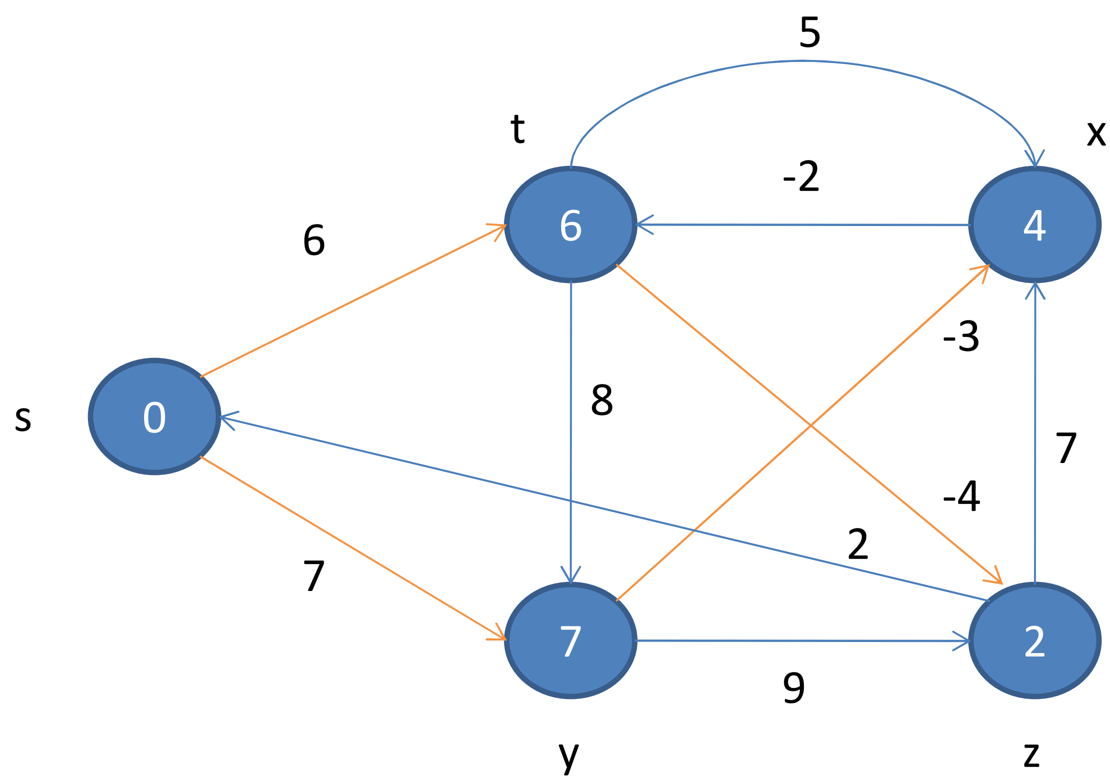
# Example: Bellman-Ford



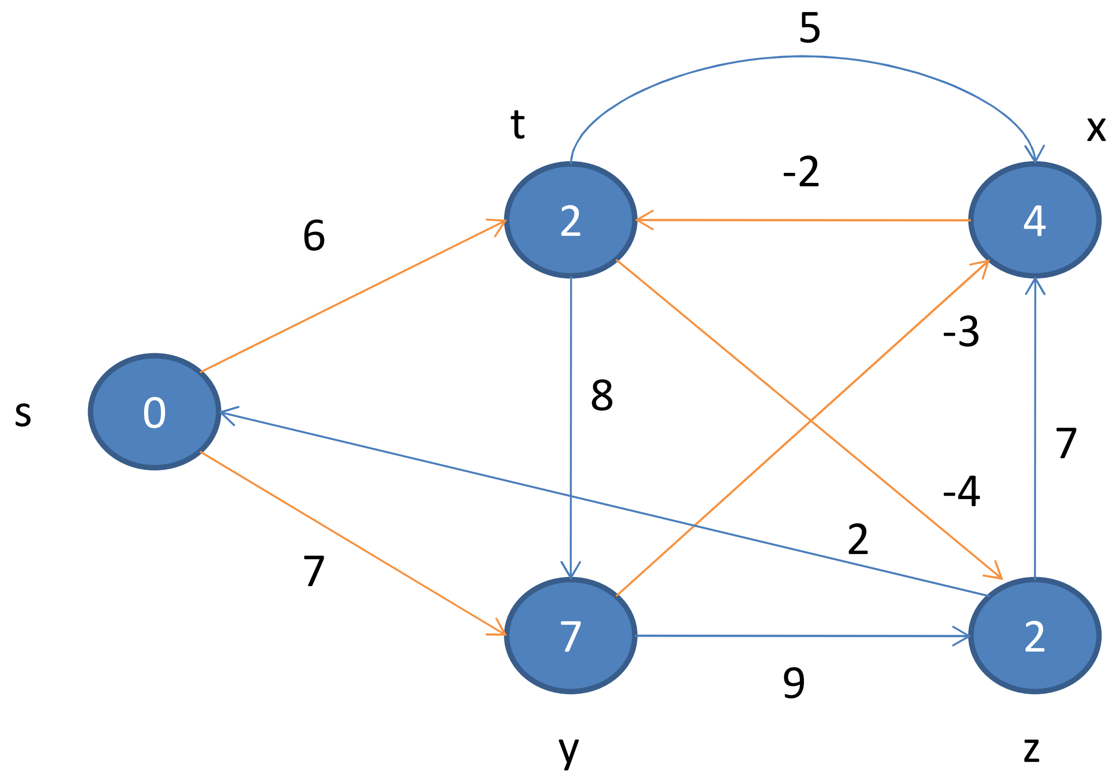
# Example: Bellman-Ford



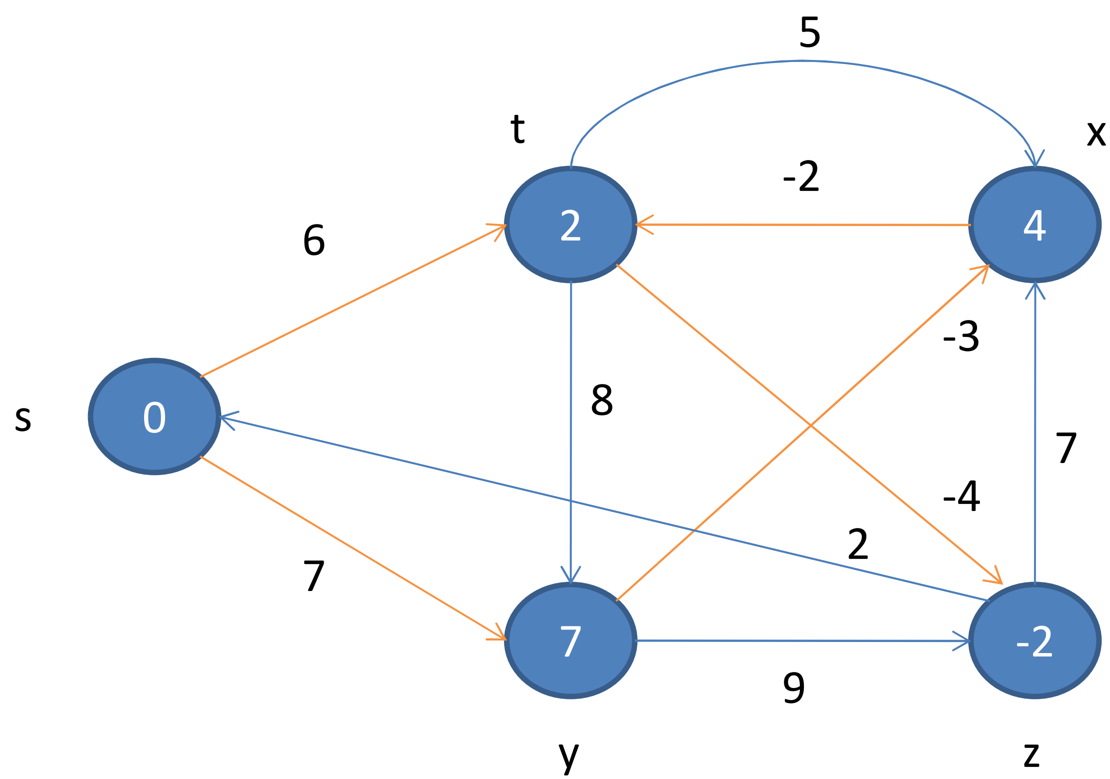
# Example: Bellman-Ford



# Example: Bellman-Ford

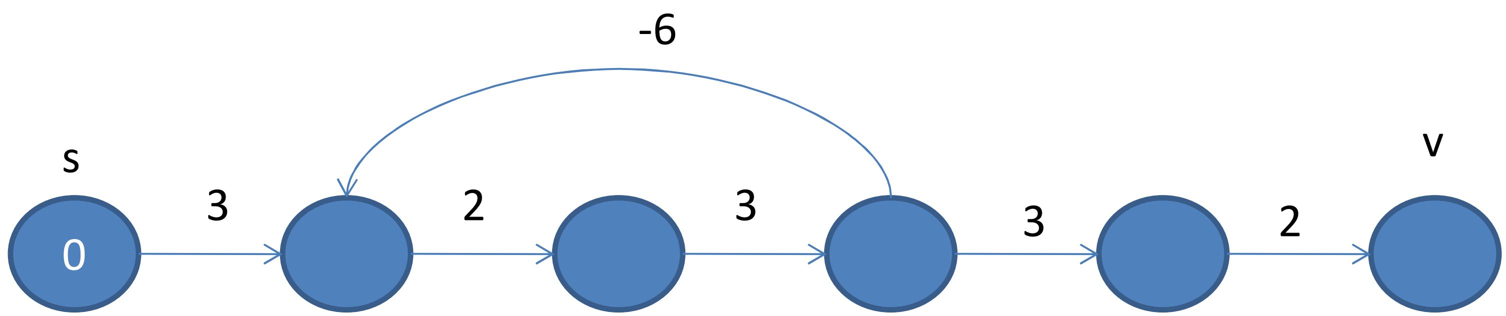


# Example: Bellman-Ford




RETURN true
With shortest path 0, 2, 4, 7, -2

# Negative weighted cycle




The shortest simple path to reach v from s = 13
If we have negative edge cycle in a path , it takes **exponential** running time to solve.