Chapter24. Shortest Path Problems

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:

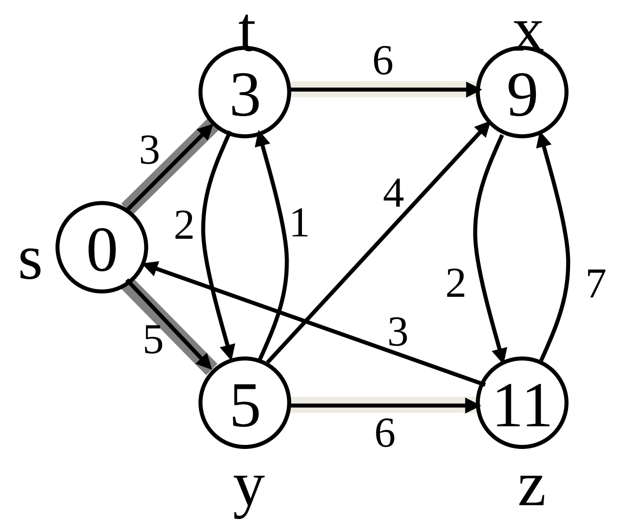
    vertices = cities

    edges = road segments between cities

    edge weights = road distances
  - Goal: find a shortest path between two vertices (cities)
- Application : Network routing, driving direction, ….

Shortest Path Problem

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



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

- **Shortest-path weight** from u to v:

$$\delta(u, v) = \begin{cases} \min\{w(p) : u \overset{p}{\rightsquigarrow} v\} & \text{if there is a path from } u \text{ to } v \\ \infty & \text{otherwise} \end{cases}$$

Note: there might be <u>multiple shortest</u> paths from u to v

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

Variants of Shortest Paths

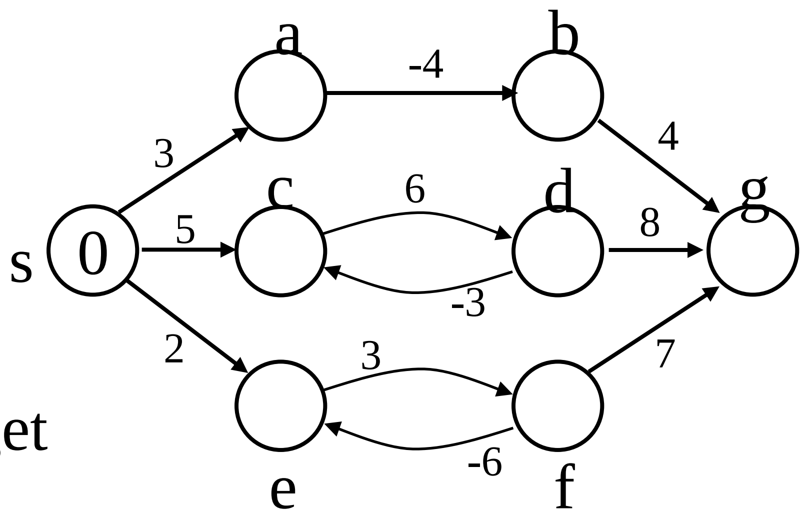
- **Single-pair shortest path**
  - Find a shortest path from u to v for given vertices u and v
- **All-pairs shortest-paths**
  - Find a shortest path from u to v for every pair of vertices u and v

Variants of Shortest Paths

Single-Source Single-Destination (1-1) -No good solution that beats 1-M variant -This problem is mapped to the 1-M variant	**Single-Source All-Destination (1-M)** -Need to be solved (several algorithms)
All-Sources Single-Destination (M-1) -Reverse all edges in the graph -This also is mapped to the (1-M) variant	**All-Sources All-Destinations (M-M)** -Need to be solved (several algorithms) -We will skip it

Negative-Weight Edges

- Negative-weight edges may form negative-weight cycles
- If such cycles are reachable from the source, then δ(s, v) is not properly defined!
  - Keep going around the cycle, and get $w(s, v) = -\infty$ for all v on the cycle



Negative-Weight Edges

- $s \to a$: only one path

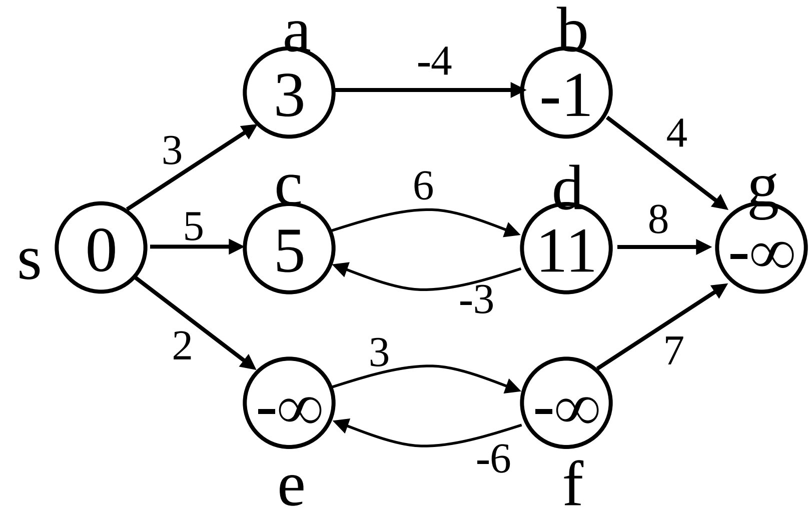
  $\delta(s, a) = w(s, a) = 3$
- $s \to b$: only one path

  $\delta(s, b) = w(s, a) + w(a, b) = -1$
- $s \to c$: infinitely many paths

  $\langle s, c\rangle, \langle s, c, d, c\rangle, \langle s, c, d, c, d, c\rangle$

  cycle has positive weight (6 - 3 = 3)

  $\langle s, c\rangle$ is shortest path with weight $\delta(s, b) = w(s, c) = 5$

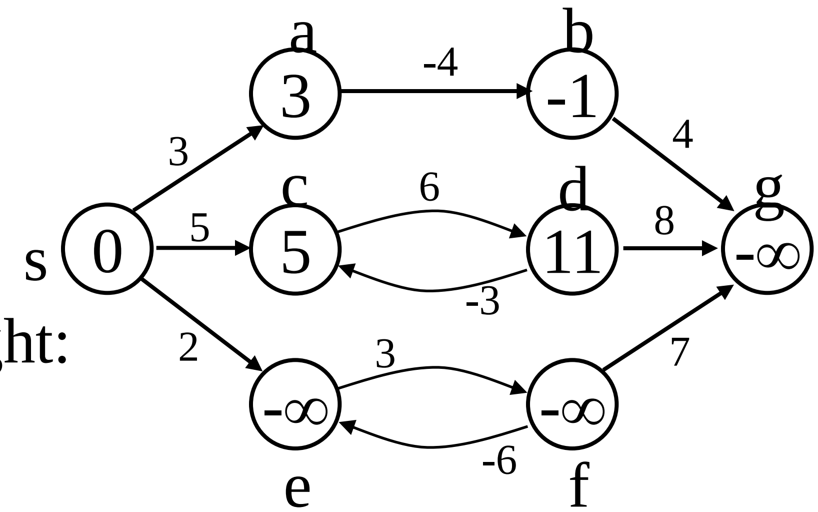


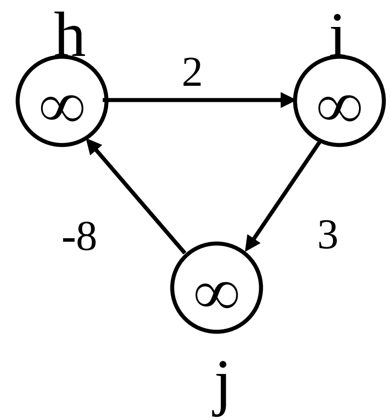
Negative-Weight Edges

- $s \rightarrow e$: infinitely many paths:
  - $\langle s, e\rangle, \langle s, e, f, e\rangle, \langle s, e, f, e, f, e\rangle$
  - cycle $\langle e, f, e\rangle$ has negative weight:

    $3 + (-6) = -3$
  - can find paths from **s** to **e** with arbitrarily large negative weights
  - $\delta(s, e) = -\infty \Rightarrow$ no shortest path exists between **s** and **e**
  - Similarly: $\delta(s, f) = -\infty$,

    $\delta(s, g) = -\infty$





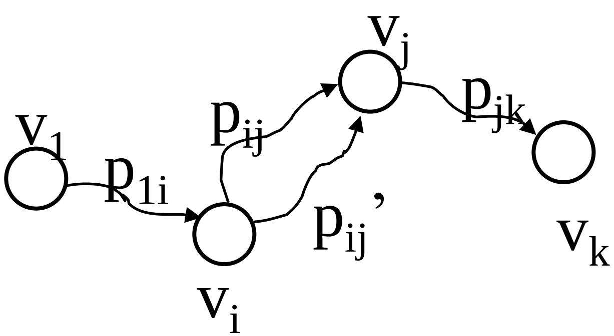
h, i, j not reachable from s

$\delta(s, h) = \delta(s, i) = \delta(s, j) = \infty$

Cycles

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

Optimal Substructure Theorem



Given:

- A weighted, directed graph $G = (V, E)$
- A weight function $w: E \rightarrow R$,
- A shortest path $p = \langle v_1, v_2, \ldots, v_k \rangle$ from v_1 to v_k
- A subpath of p: $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$, with $1 \le i \le j \le k$

Then: p_{ij} is a shortest path from v_i to v_j

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

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

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

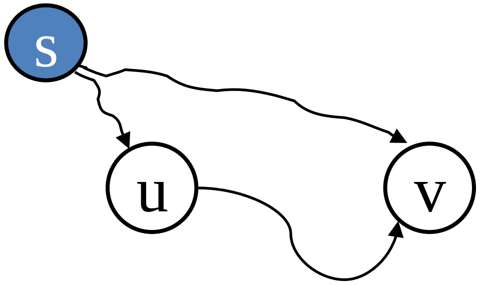
$\Rightarrow w(p') = w(p_{1i}) + w(p_{ij}') + w(p_{jk}) < w(p)$

<u>contradiction!</u>

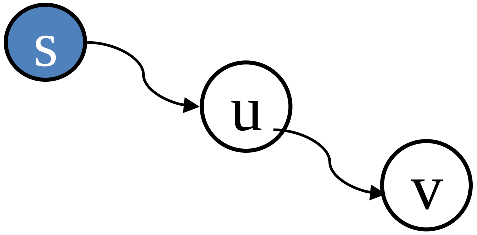
Triangle Inequality

For all $(u, v) \in E$, we have:

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



- If u is on the shortest path to v, we have the equality sign



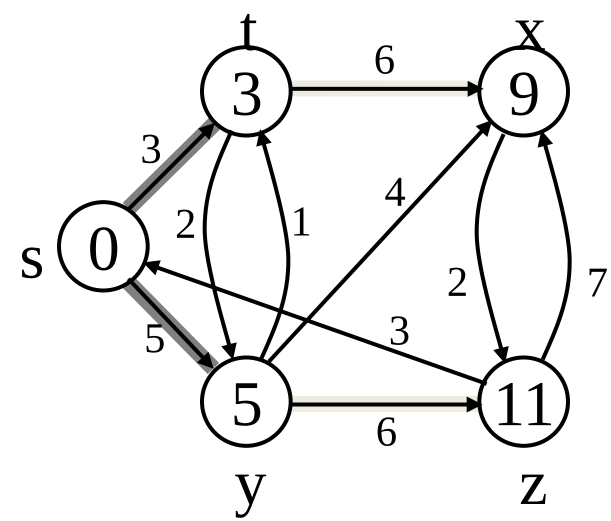
Algorithms

- 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

Shortest-Paths Notation

For each vertex $v \in V$:

- $\delta(s, v)$: **shortest-path weight**
- $d[v]$: shortest-path weight **estimate**
  - 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**



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

Relaxation

- Given weighted graph G = (V, E) with source node $s \in V$ and other node $v \in V$ ($v = s$), we'll maintain d[v], which is upper bound on (s, v)
- Relaxation of an edge (u, v) is the process of testing whether we can decrease d[v], yielding a tighter upper bound.

Relaxation Step

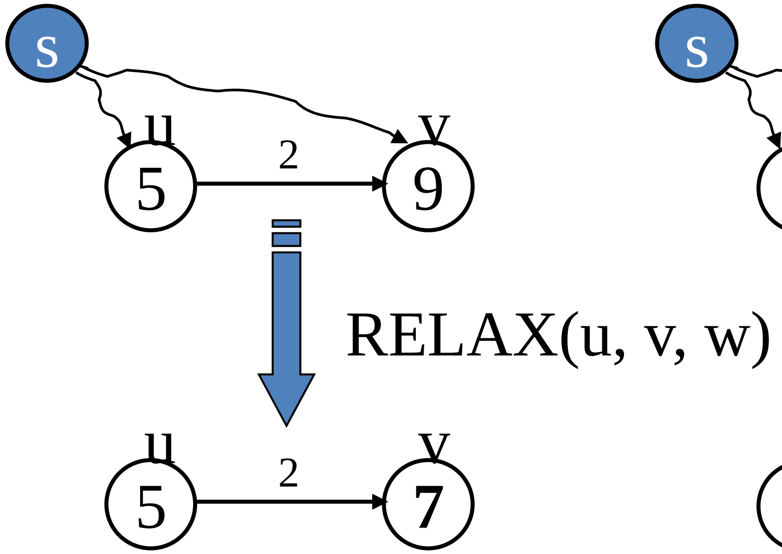
- **Relaxing** an edge (u, v) = testing whether we can improve the shortest path to v found so far by going through u

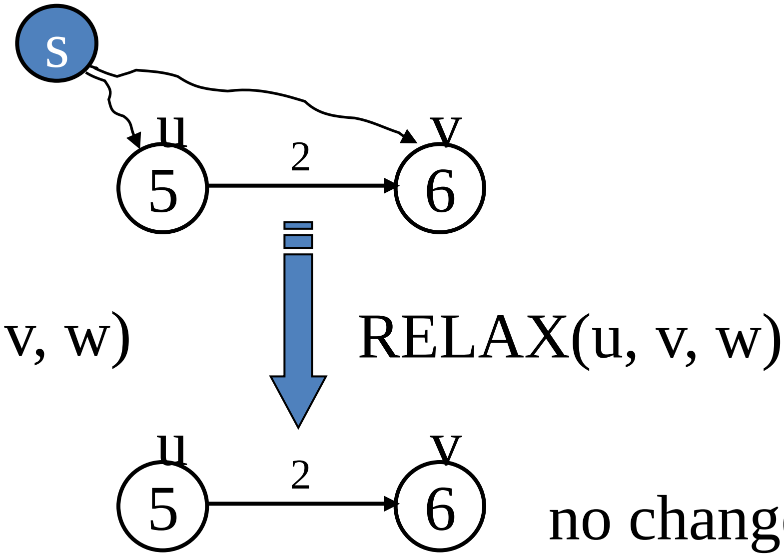
  If $d[v] > d[u] + w(u, v)$

  we can improve the shortest path to v

  $\Rightarrow d[v]=d[u]+w(u,v)$

  $\Rightarrow \pi[v] \leftarrow u$





no change!

Bellman-Ford Algorithm

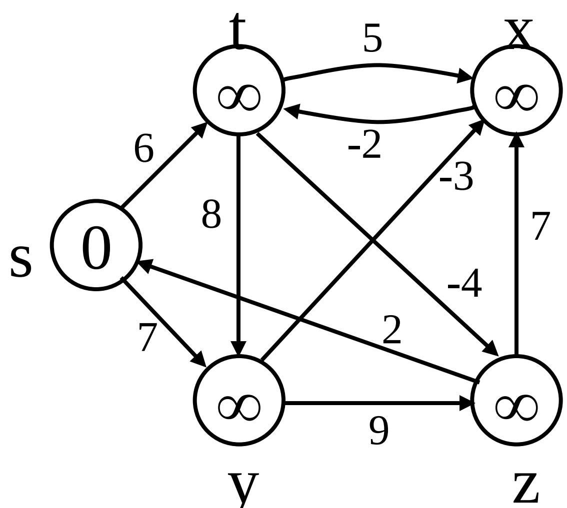
- Single-source shortest path problem
  - Computes $\delta(s, v)$ and $\pi[v]$ for all $v \in V$

- Works with negative-weight edges and detects if there is a negative-weight cycle.
  - Returns TRUE if no negative-weight cycles are reachable from the source s
  - Returns FALSE otherwise $\Rightarrow$ no solution exists

Bellman-Ford Algorithm (cont'd)

- Idea:
  - 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)

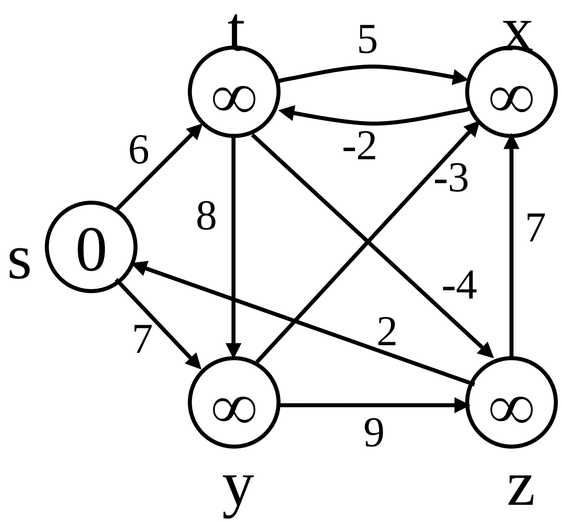


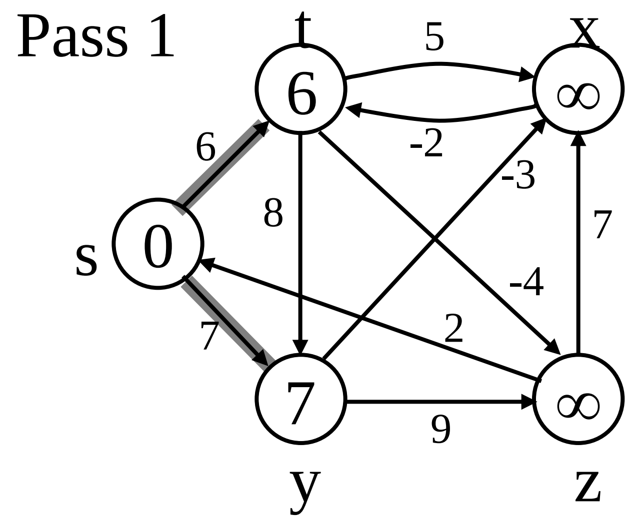
Bellman-Ford(G, w, s)

```
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |V| − 1 do
    for each edge (u, v) ∈ E do
        RELAX(u, v, w)
    end
end
for each edge (u, v) ∈ E do
    if d[v] > d[u] + w(u, v) then
        return FALSE // G has a negative-wt cycle

end
return TRUE // G has no neg-wt cycle reachable frm s
```

Bellman-Ford(G, w, s)



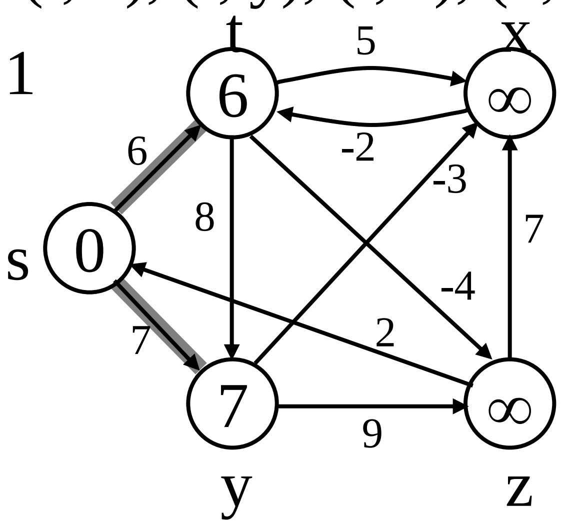


E: (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)

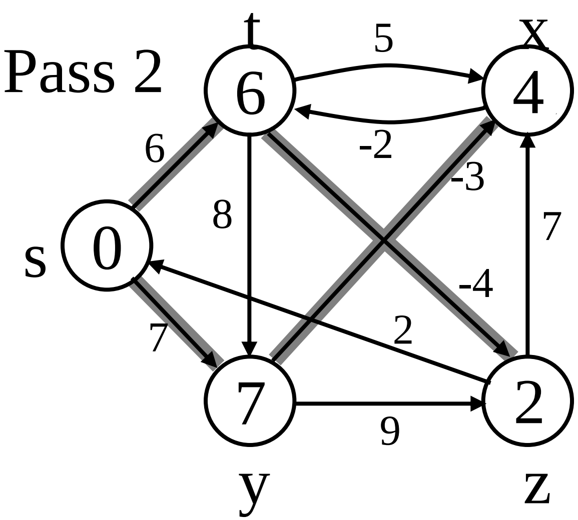
Example

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

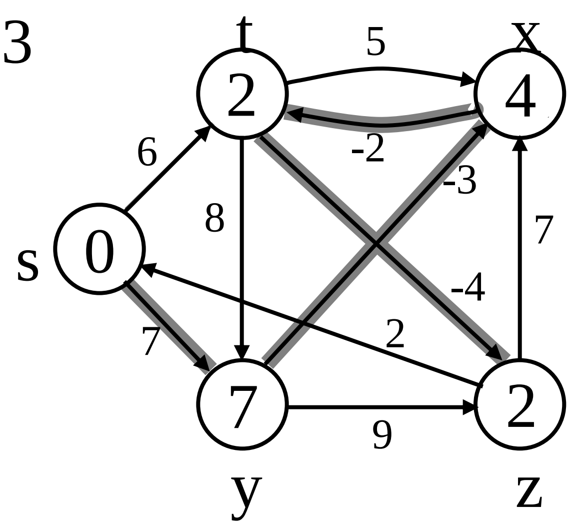
Pass 1



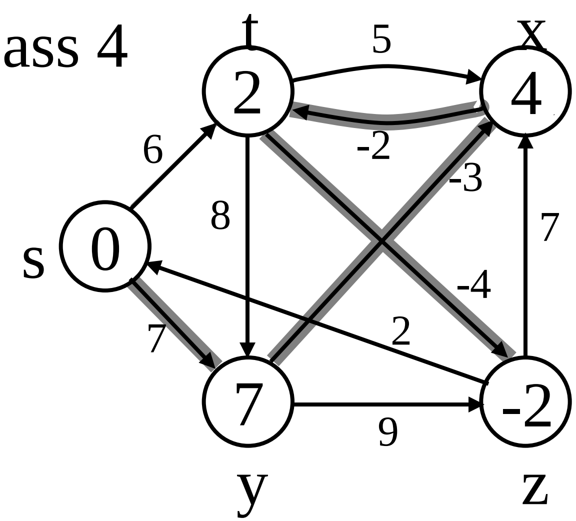
Pass 2



Pass 3

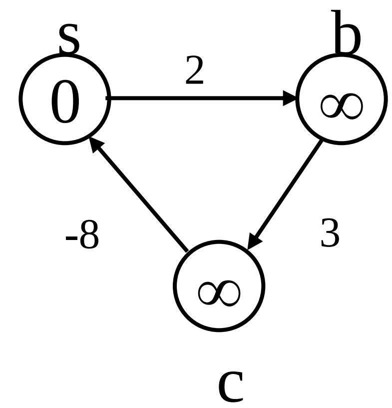


Pass 4

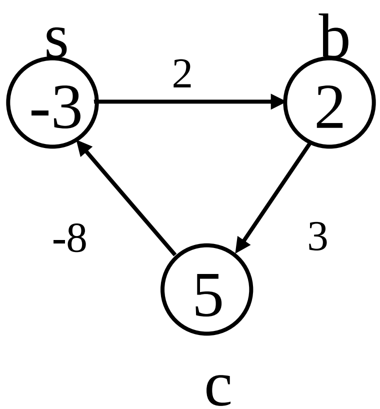


Detecting Negative Cycles (perform extra test after V-1 iterations)

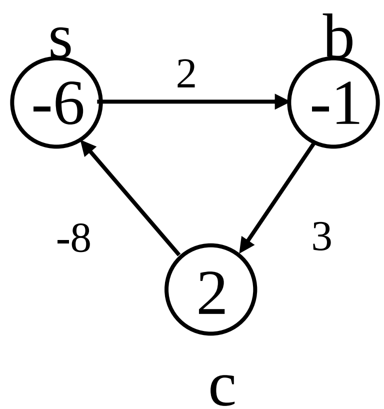
- **for** each edge $(u, v) \in E$
- **do if** $d[v] > d[u] + w(u, v)$
- **then return** FALSE
- **return** TRUE



1st pass



2nd pass



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

Look at edge (s, b):

$d[b] = -1$

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

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

Time Complexity of Bellman-Ford Algorithm

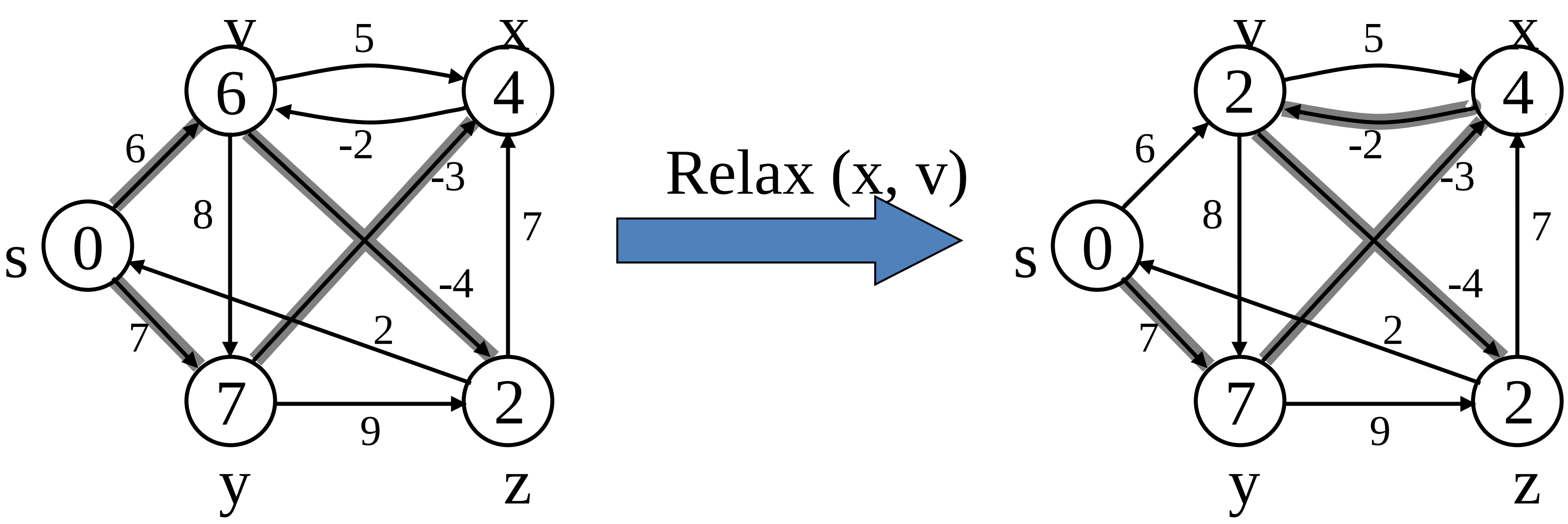
```
Initialize-Single-Source(G, s)                  Θ(V)
for i = 1 to |V| − 1 do                          O(V)
    for each edge (u, v) ∈ E do                  O(E)     O(VE)
        Relax(u, v, w)
    end
end
for each edge (u, v) ∈ E do                      O(E)
    if d[v] > d[u] + w(u, v) then
        return FALSE // G has a negative-wt cycle

end
return TRUE // G has no neg-wt cycle reachable frm s
```

Running time: $O(V+VE+E)=O(VE)$

Shortest Path Properties

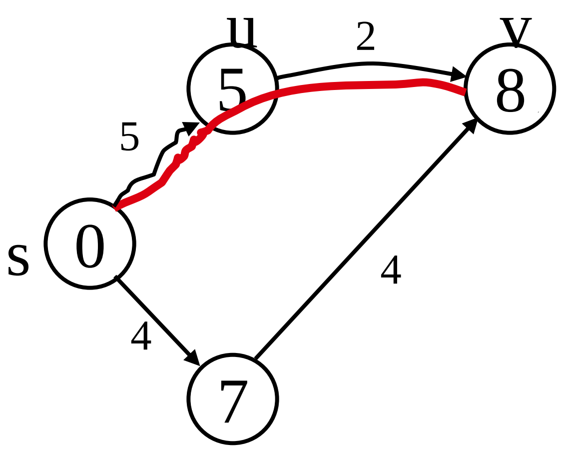
- **Upper-bound property**
  - We always have $d[v] \geq \delta(s, v)$ for all v.
  - The estimate never goes up : relaxation only lowers the estimate



Shortest Path Properties

- **Convergence property**

  If $s \leadsto u \rightarrow v$ is a shortest path, and if $d[u] = \delta(s, u)$ at any time prior to relaxing edge (u, v), then $d[v] = \delta(s, v)$ at all times after relaxing (u, v).



- If $d[v] > \delta(s, v) \Rightarrow$ after relaxation:

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

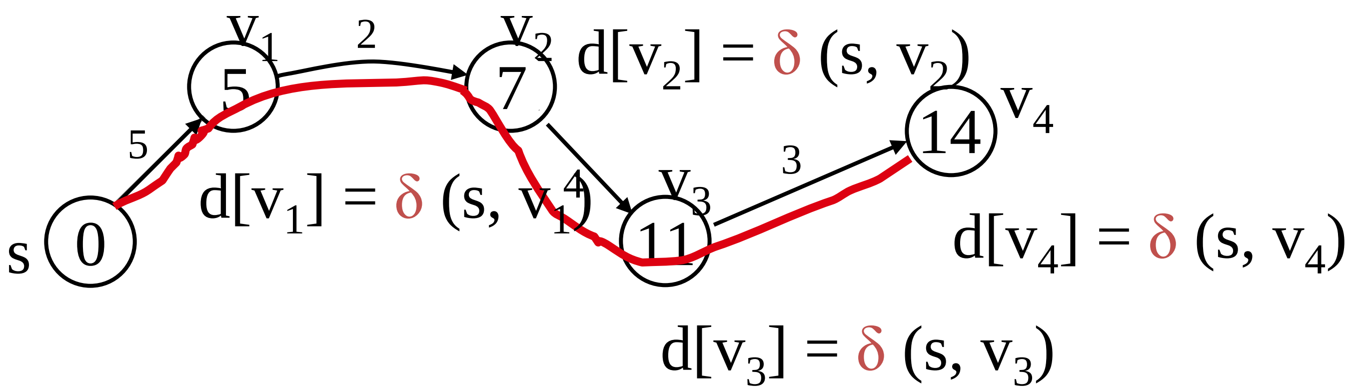
  $d[v] = 5 + 2 = 7$

- Otherwise, the value remains unchanged, because it must have been the shortest path value

Shortest Path Properties

- **Path relaxation property**

Let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path from $s = v_0$ to v_k. If we relax, in order, $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, even intermixed with other relaxations, then $d[v_k] = \delta(s, v_k)$.



Correctness of Belman-Ford Algorithm

- **Theorem:** Show that $d[v]= \delta (s, v)$, for every v, after $|V-1|$ passes.

<u>Case 1:</u> G does not contain negative cycles which are reachable from s

- Assume that the shortest path from s to v is

  $p = \langle v_0, v_1, . . . , v_k\rangle$, where $s=v_0$ and $v=v_k$, $k\leq|V-1|$

- Use mathematical induction on the number of

passes i to show that:

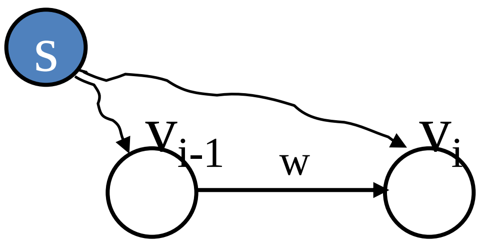
$$d[v_i]= \delta (s, v_i) , i=0,1,\ldots,k$$

Correctness of Belman-Ford Algorithm

Base Case: i=0 $d[v_0]= \delta (s, v_0)= \delta (s, s)= 0$

Inductive Hypothesis: $d[v_{i-1}]= \delta (s, v_{i-1})$

Inductive Step: $d[v_i]= \delta (s, v_i)$



$d[v_{i-1}]= \delta (s, v_{i-1})$

After relaxing (v_{i-1}, v_i):

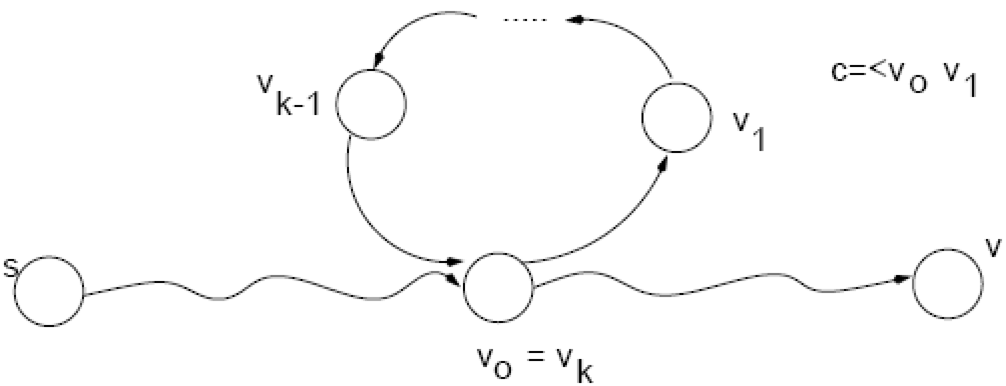
$d[v_i] \leq d[v_{i-1}]+w= \delta (s, v_{i-1})+w= \delta (s, v_i)$

From the upper bound property: $d[v_i] \geq \delta (s, v_i)$

Therefore, $d[v_i]=\delta (s, v_i)$

Correctness of Belman-Ford Algorithm

- Case 2: G contains a negative cycle which is reachable from s



c=<v_0 v_1 v_k> is a negative cycle

$$\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$$

Proof by Contradiction: suppose the algorithm returns a solution

After relaxing (v_{i-1}, v_i): $\mathbf{d}[v_i] \leq \mathbf{d}[v_{i-1}] + w(v_{i-1}, v_i)$

or $\sum_{i=1}^{k} \mathbf{d}[v_i] \leq \sum_{i=1}^{k} \mathbf{d}[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i)$

or $\sum_{i=1}^{k} w(v_{i-1}, v_i) \geq 0$ $(\sum_{i=1}^{k} \mathbf{d}[v_i] = \sum_{i=1}^{k} \mathbf{d}[v_{i-1}])$

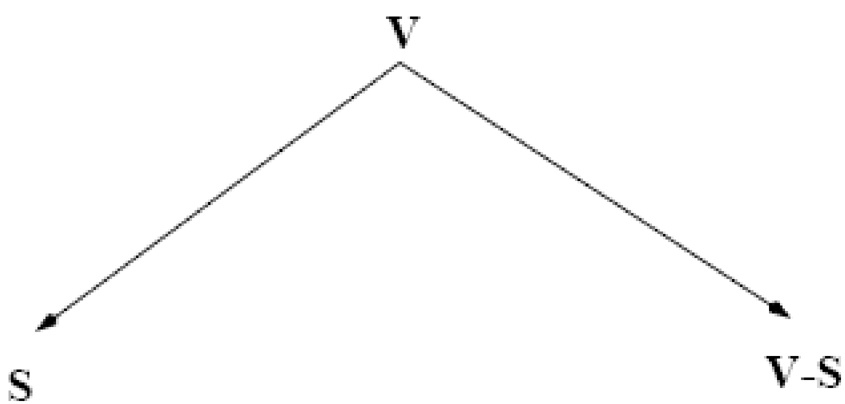
Contradiction!

Dijkstra's Algorithm

- Greedy algorithm
- Faster than Bellman-Ford
- Requires all edge weights to be nonnegative
- Maintains set S of vertices whose final shortest path weights from s have been determined
- Uses <u>min-priority queue</u> to repeatedly make greedy choice

Dijkstra’s Algorithm

- Single-source shortest path problem:
  - No negative-weight edges: $w(u, v) > 0, \ \forall (u, v) \in E$
- Each edge is relaxed **only once!**
- Maintains two sets of vertices:



$d[v]=\delta(s, v)$ $d[v]>\delta(s, v)$

Dijkstra's Algorithm (cont.)

- Vertices in V – S reside in a min-priority queue
  - Keys in Q are estimates of shortest-path weights d[u]
- Repeatedly select a vertex u $\in$ V – S, with the minimum shortest-path estimate d[u]
- Relax all edges leaving u

- **Steps**

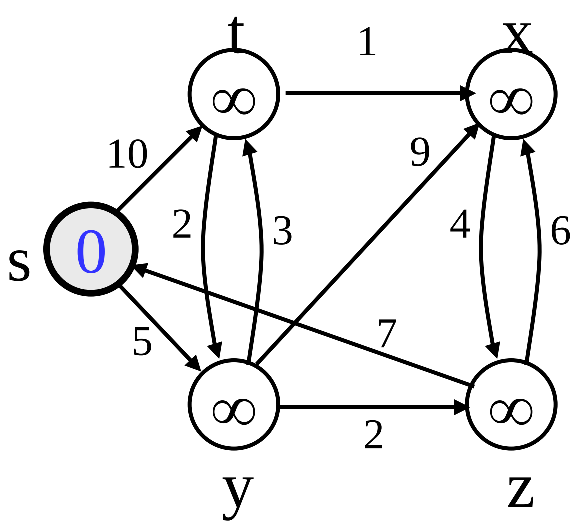
  1) Extract a vertex u from Q
  2) Insert u to S
  3) Relax all edges leaving u
  4) Update Q

Dijkstra(G, w, s)

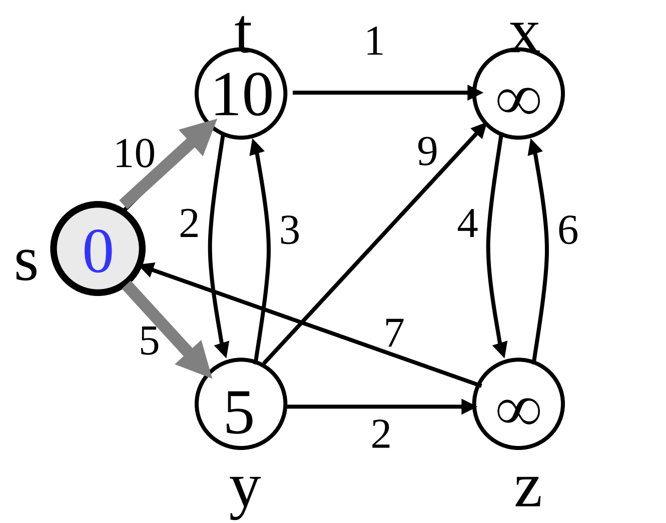
```
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = V
while Q ≠ ∅ do
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each v ∈ Adj[u] do
        RELAX(u, v, w)
    end
end
```

Dijkstra (G, w, s)

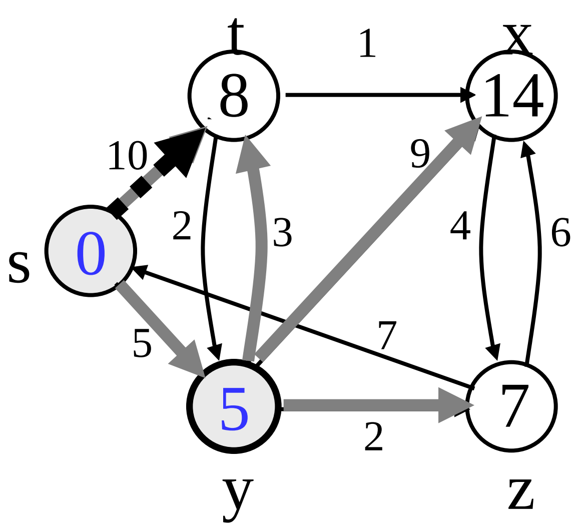
S=< > Q=<s,t,x,z,y>



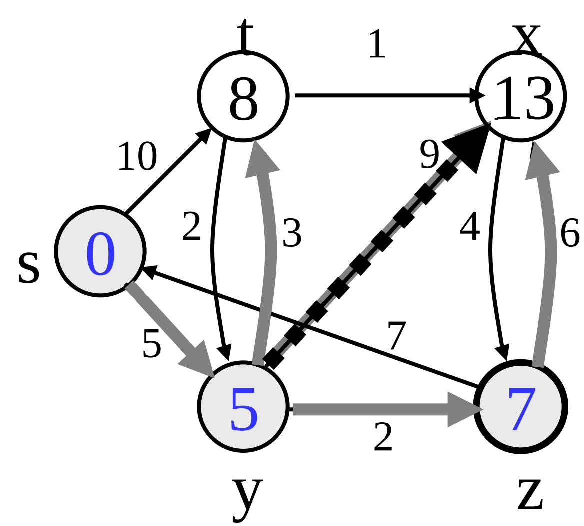
S=<s> Q=<y,t,x,z>



Example (cont.)



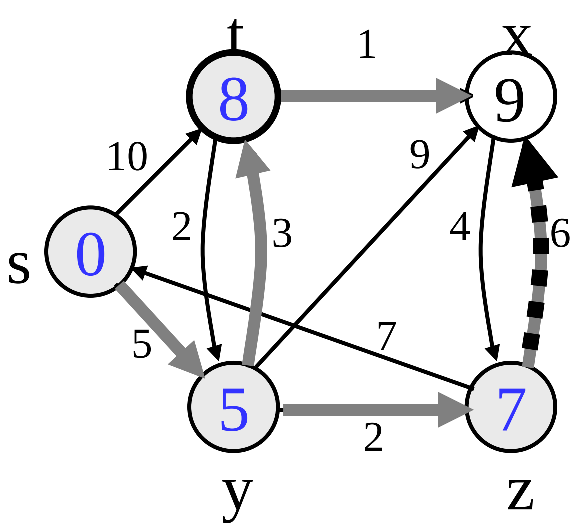
S=<s,y> Q=<z,t,x>



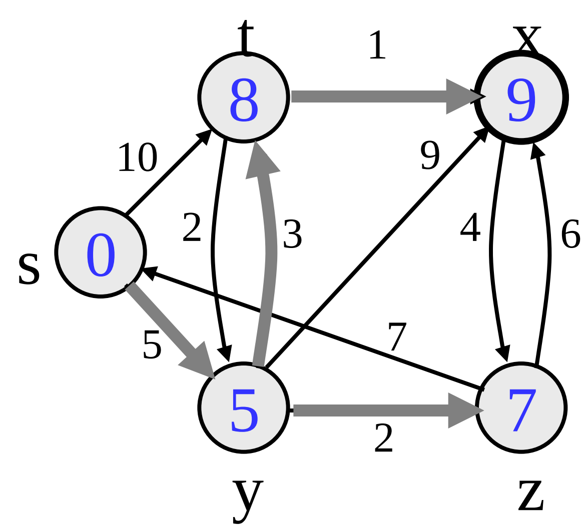
S=<s,y,z> Q=<t,x>

Example (cont.)

S=<s,y,z,t> Q=<x>



S=<s,y,z,t,x> Q=< >



Dijkstra(G, w, s)

```
INITIALIZE-SINGLE-SOURCE(G, s)      ← Θ(V)
S = ∅
Q = V                               ← O(V) build min-heap
while Q ≠ ∅ do                      ← Executed O(V) times
    u = EXTRACT-MIN(Q)              ← O(logV)
    S = S ∪ {u}                                           } O(VlgV)
    for each v ∈ Adj[u] do
        RELAX(u, v, w)                                    } O(ElgV)
    end
end
```

Running time: O(VlogV + ElogV) = O(ElogV)