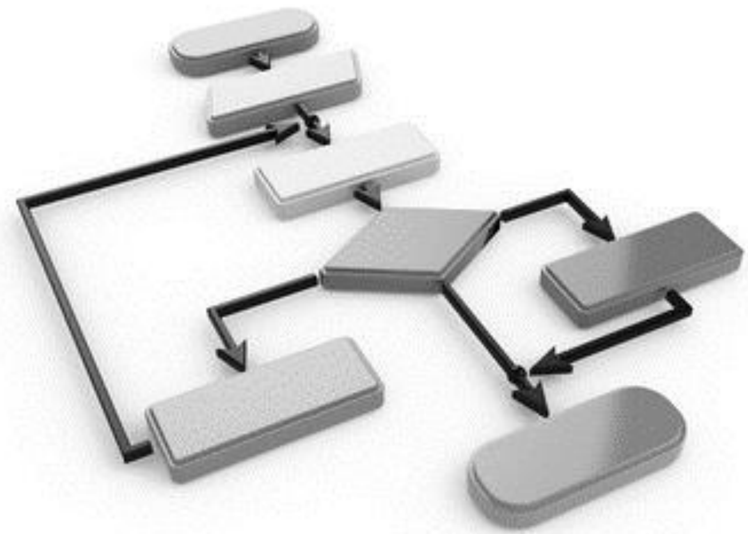


Lecture 16

Graph-Based Algorithms

CSE373: Design and Analysis of Algorithms

Shortest Path Problems

Modeling problems as graph problems:

Road map is a weighted graph:

vertices = cities

edges = road segments between cities

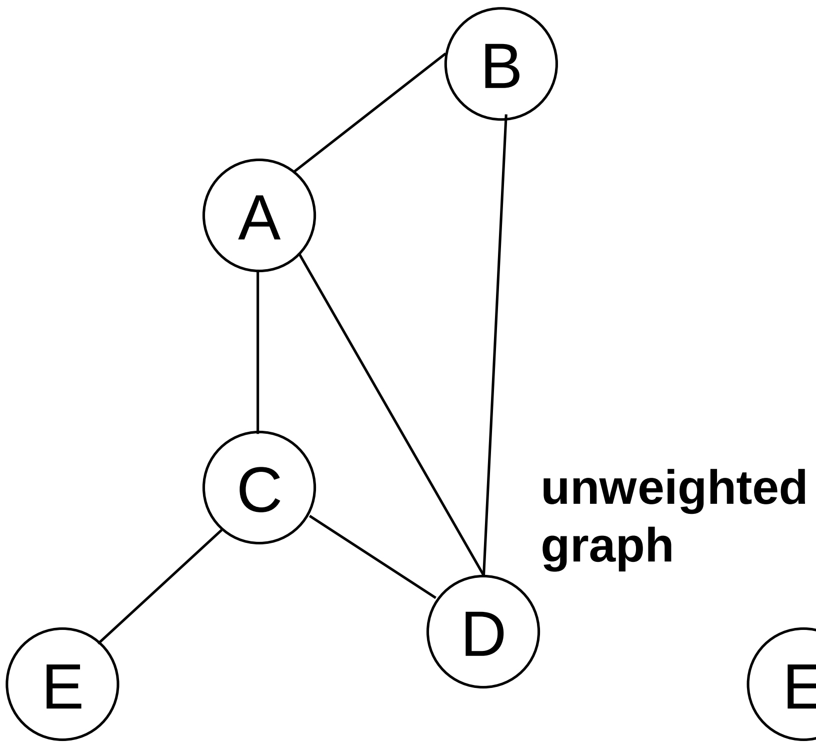
edge weights = road distances

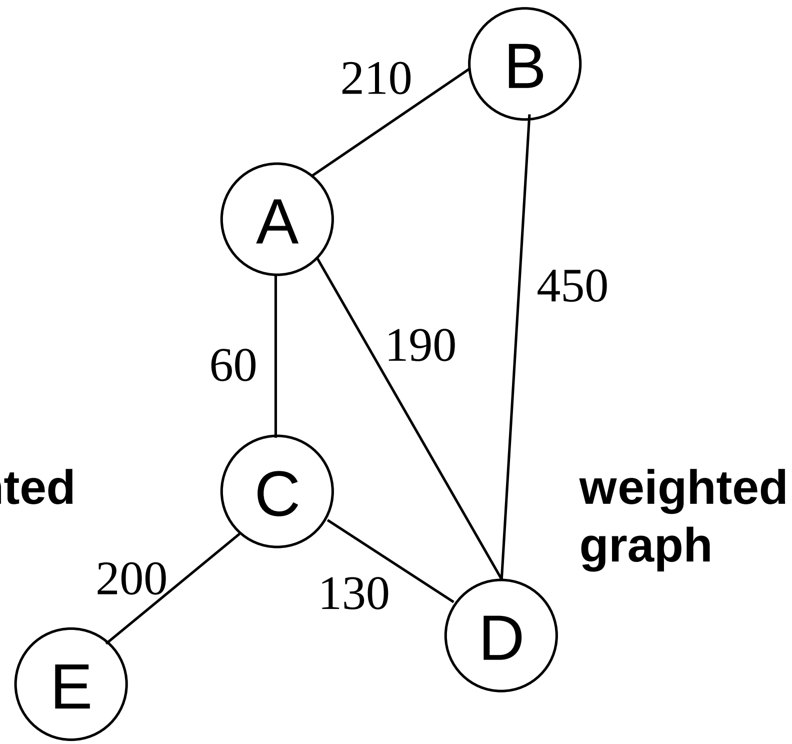
Goal: find a shortest path between two vertices (cities)

Shortest Path Problems

What is shortest path ?

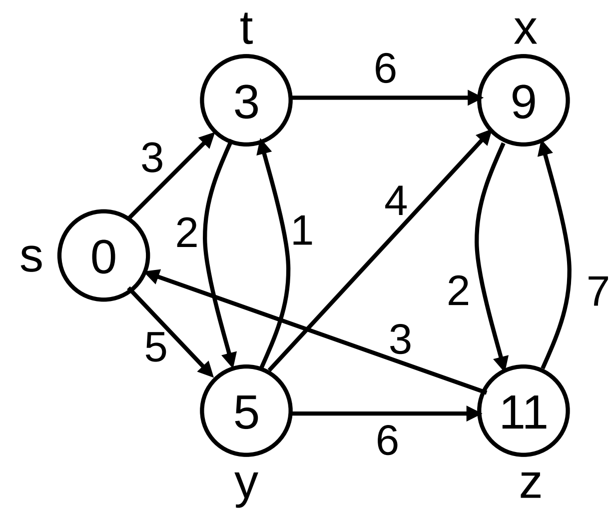
- shortest length between two vertices for an unweighted graph:
- smallest cost between two vertices *for* a weighted graph:





Shortest Path Problems

- **Input:**
  - Directed graph $G = (V, E)$
  - Weight function $w : E \rightarrow \mathbf{R}$



- **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:

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

Variants of Shortest Paths

Single-source shortest path

Given G = (V, E), find a shortest path from a given source vertex *s* to each vertex $v \in V$

Single-destination shortest path

Find a shortest path to a given destination vertex **t** from each vertex v

Reverse 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

Solve the single-source problem

All-pairs shortest-paths

Find a shortest path from u to v for every pair of vertices u and v

Shortest-Path Representation

For each vertex $v \in V$:

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

- Initially, $d[v]=\infty$
- Reduces as algorithms progress

$\pi[v]$ = **predecessor** of v on a shortest path from s

- If no predecessor, $\pi[v]$ = NIL
- π induces a tree—**shortest-path tree**

Shortest paths & shortest path trees are not unique

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

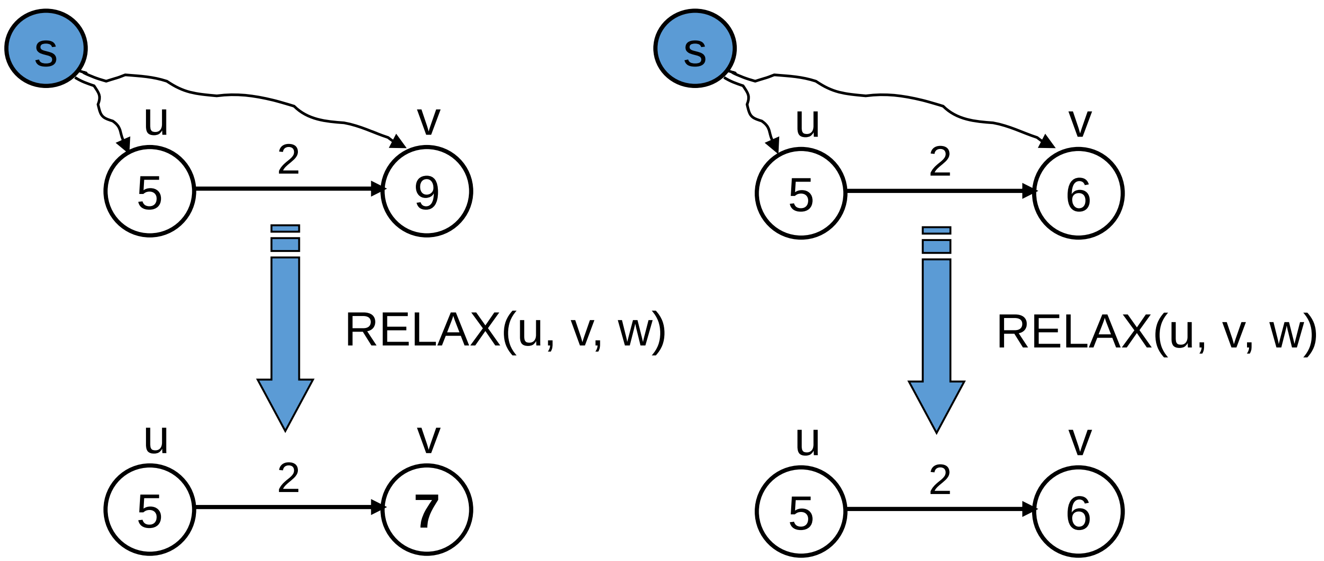
Relaxation

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

$\Rightarrow$ update $d[v]$ and $\pi[v]$



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

RELAX(u, v, w)

1. **if** d[v] > d[u] + w(u, v)
2. **then** d[v] $\leftarrow$ d[u] + w(u, v)
3. π[v] $\leftarrow$ u

All the single-source shortest-paths algorithms

start by calling INIT-SINGLE-SOURCE

then relax edges

The algorithms differ in the order and how many times they relax each edge

Dijkstra’s Algorithm

Single-source shortest path problem:

- No negative-weight edges: $w(u, v) > 0 \ \forall \ (u, v) \in E$

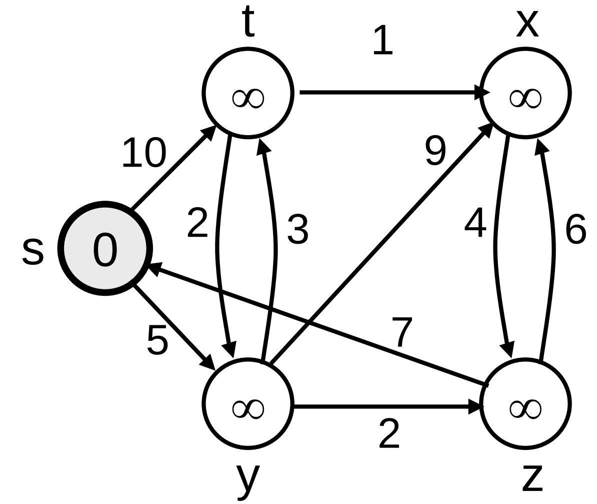
Maintains two sets of vertices:

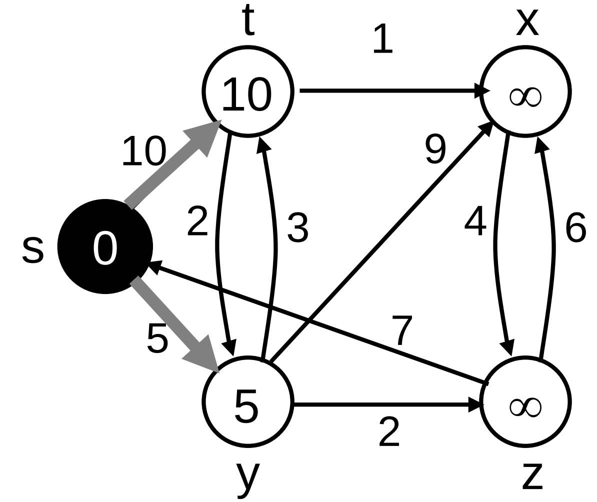
- S = vertices whose final shortest-path weights have already been determined
- Q = vertices in V – S: min-priority queue
  - Keys in Q are estimates of shortest-path weights (d[v])

Repeatedly select a vertex $u \in V - S$, with the minimum shortest-path estimate d[v]

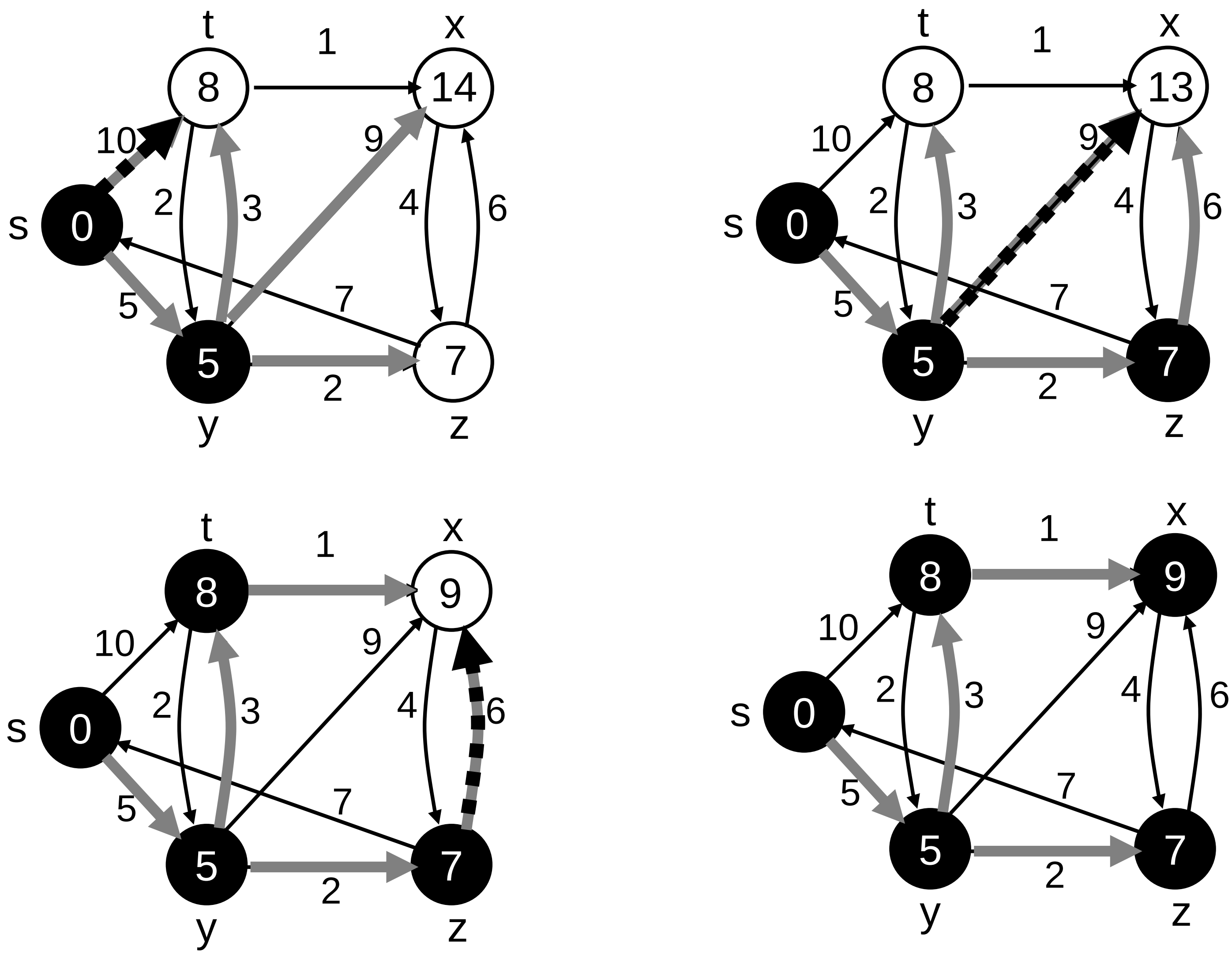
Dijkstra (G, w, s)

```
INITIALIZE-SINGLE-SOURCE(V, 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)
```





Example


t x s y z
8 14 0 5 7
1 10 2 3 9 4 6 5 7 2
t x s y z
8 13 0 5 7
1 10 2 3 9 4 6 5 7 2
t x s y z
8 9 0 5 7
1 10 2 3 9 4 6 5 7 2
t x s y z
8 9 0 5 7
1 10 2 3 9 4 6 5 7 2

Dijkstra's Pseudo Code

Graph *G*, weight function *w*, root *s*

```
DIJKSTRA(G, w, s)
for each v ∈ V
        do d[v] ← ∞
d[s] ← 0
S ← ∅    ▷ Set of discovered nodes
Q ← V
while Q ≠ ∅
        do u ← EXTRACT-MIN(Q)
           S ← S ∪ {u}
           for each v ∈ Adj[u]
                do if d[v] > d[u] + w(u, v)
                      then d[v] ← d[u] + w(u, v)
```

relaxing edges

Dijkstra (G, w, s)

1. INITIALIZE-SINGLE-SOURCE(V, s) $\longleftarrow$ $\Theta(V)$
2. S $\leftarrow \varnothing$
3. Q $\leftarrow$ V[G] $\longleftarrow$ O(V) build min-heap
4. **while** Q $\neq \varnothing$ $\longleftarrow$ Executed O(V) times
5. **do** u $\leftarrow$ EXTRACT-MIN(Q) $\longleftarrow$ O(lgV)
6. S $\leftarrow$ S $\cup$ {u}
7. **for** each vertex v $\in$ $Adj[u]$
8. **do** RELAX(u, v, w) $\longleftarrow$ O(E) times; O(lgV)

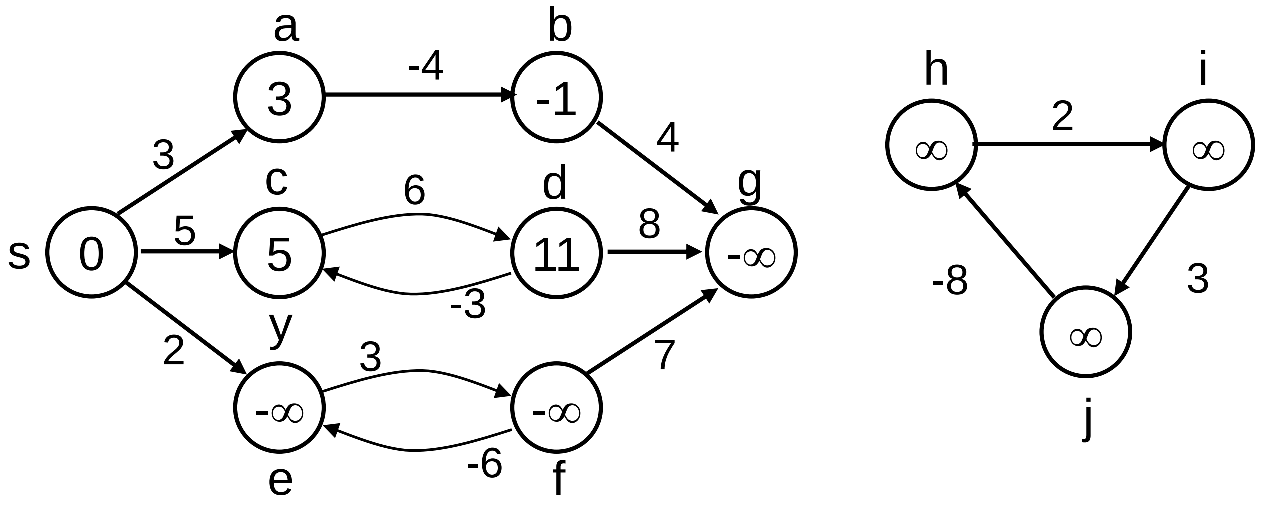
Running time: $O(V\lg V + E\lg V) = O(E\lg V)$

Dijkstra's Running Time

Q	T(Extract -Min)	T(Decrease-Key)	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$

Negative-Weight Edges

What if we have negative-weight edges?



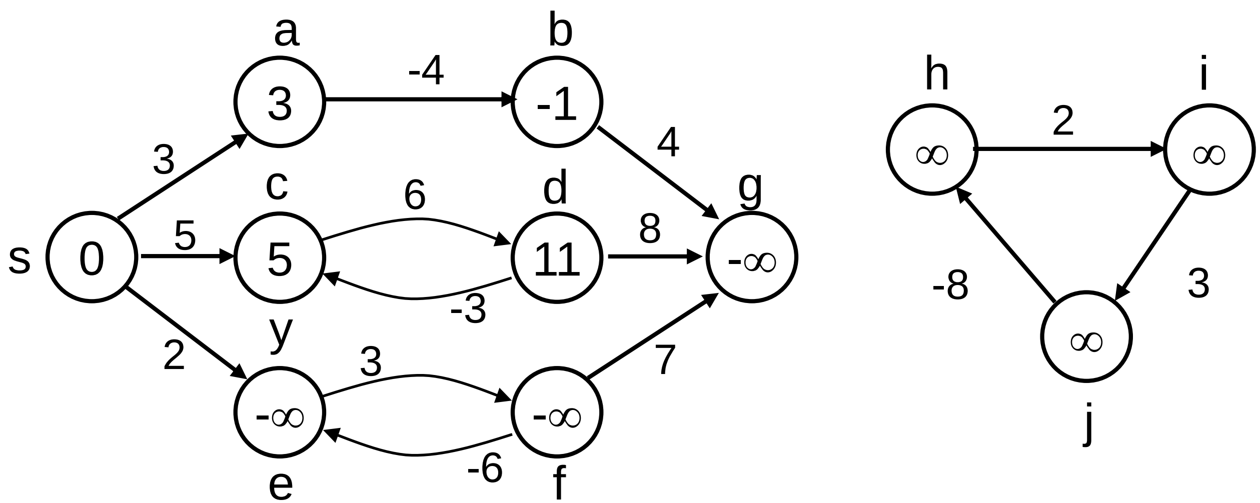
Negative-Weight Edges

$s \rightarrow a$: only one path

$\delta(s, a) = w(s, a) = 3$

$s \rightarrow b$: only one path

$\delta(s, b) = w(s, a) + w(a, b) = -1$



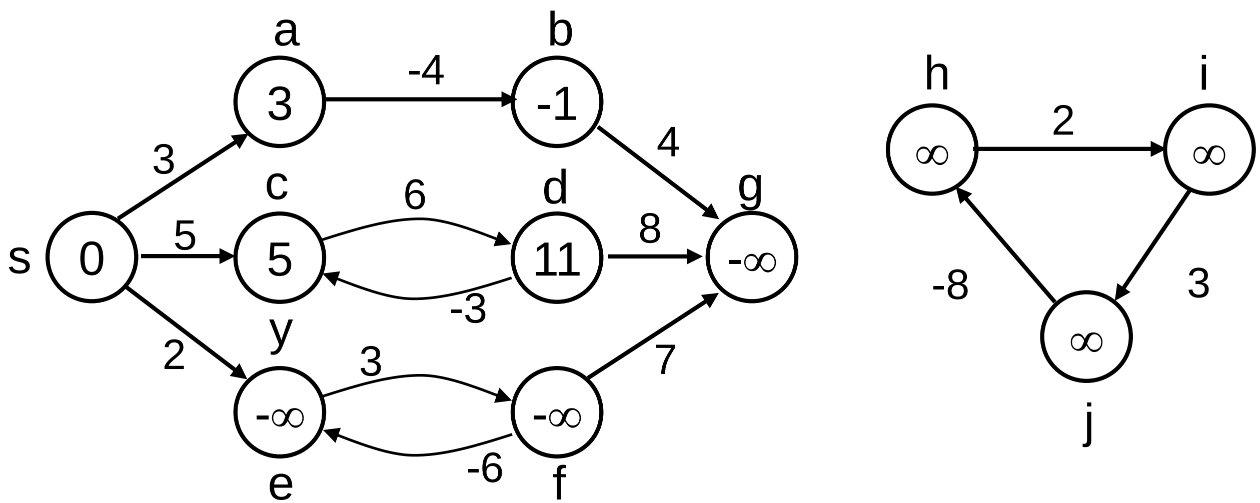
Negative-Weight Edges

s $\rightarrow$ c: infinitely many paths

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

cycle $\langle$c, d, c$\rangle$ has positive weight (6 - 3 = 3)

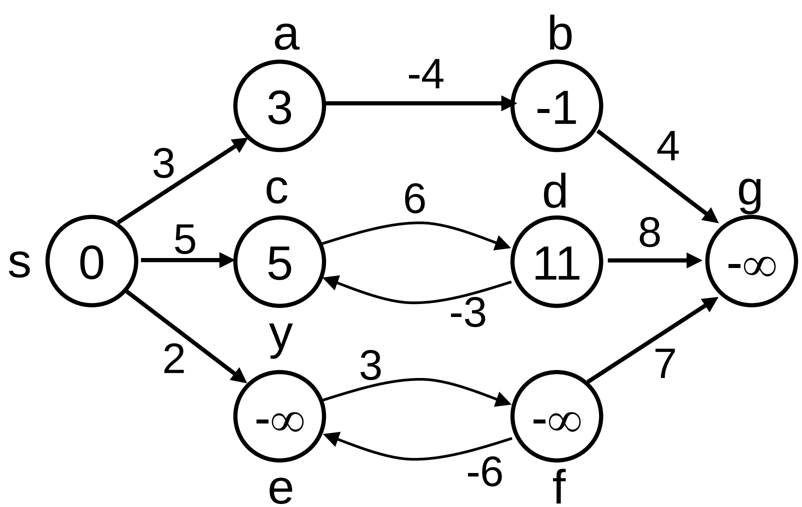
$\langle$s, c$\rangle$ is shortest path with weight δ(s, b) = w(s, c) = 5

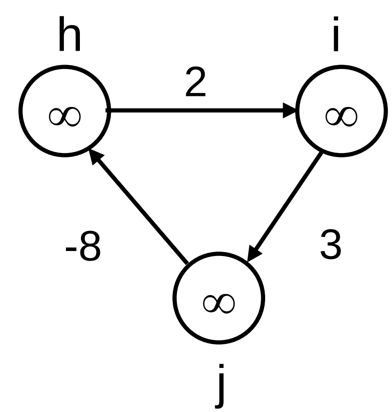


Negative-Weight Edges

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
- many paths from **s** to **e** with arbitrarily large negative weights
- δ(s, e) = - ∞ $\Rightarrow$ no shortest path exists between **s** and **e**
- Similarly: δ(s, f) = - ∞, δ(s, g) = - ∞



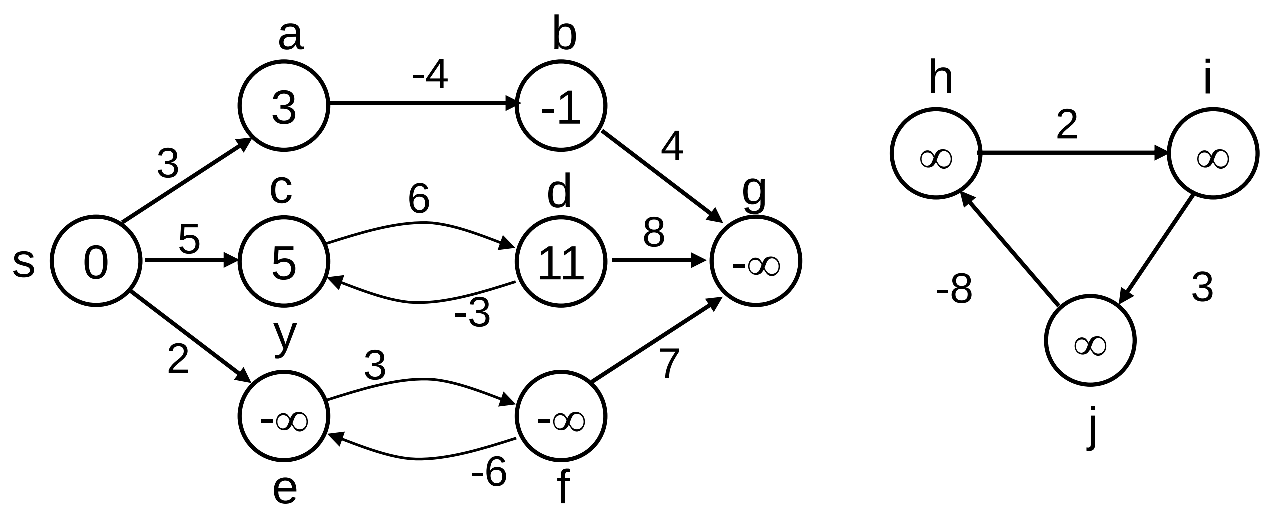


h, i, j not reachable from s

δ(s, h) = δ(s, i) = δ(s, j) = ∞

Negative-Weight Edges

- Negative-weight edges may form negative-weight cycles
- If such cycles are reachable from the source: $\delta(s, v)$ is not properly defined



Cycles

Can shortest paths contain cycles?

Negative-weight cycles No!

Positive-weight cycles: No!

By removing the cycle we can get a shorter path

We will assume that when we are finding shortest paths, the paths will have no cycles

Bellman-Ford Algorithm

Single-source shortest paths problem

- Computes d[v] and π[v] for all v $\in$ V

Allows negative edge weights

Returns:

- TRUE if no negative-weight cycles are reachable from the source s
- FALSE otherwise $\Rightarrow$ no solution exists

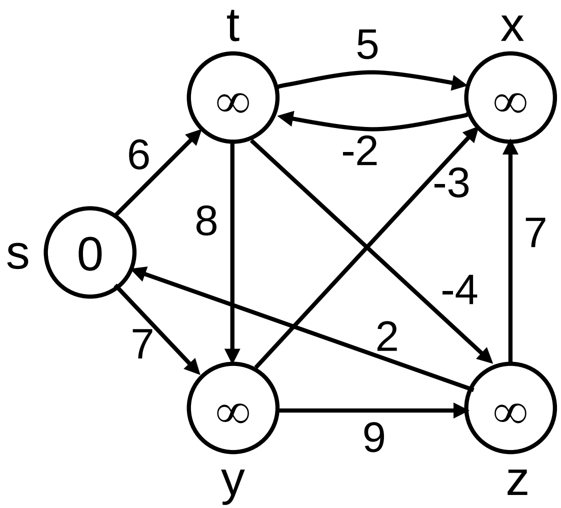
Idea:

- Traverse all the edges |V – 1| times, every time performing a relaxation step of each edge

BELLMAN-FORD(V, E, w, s)

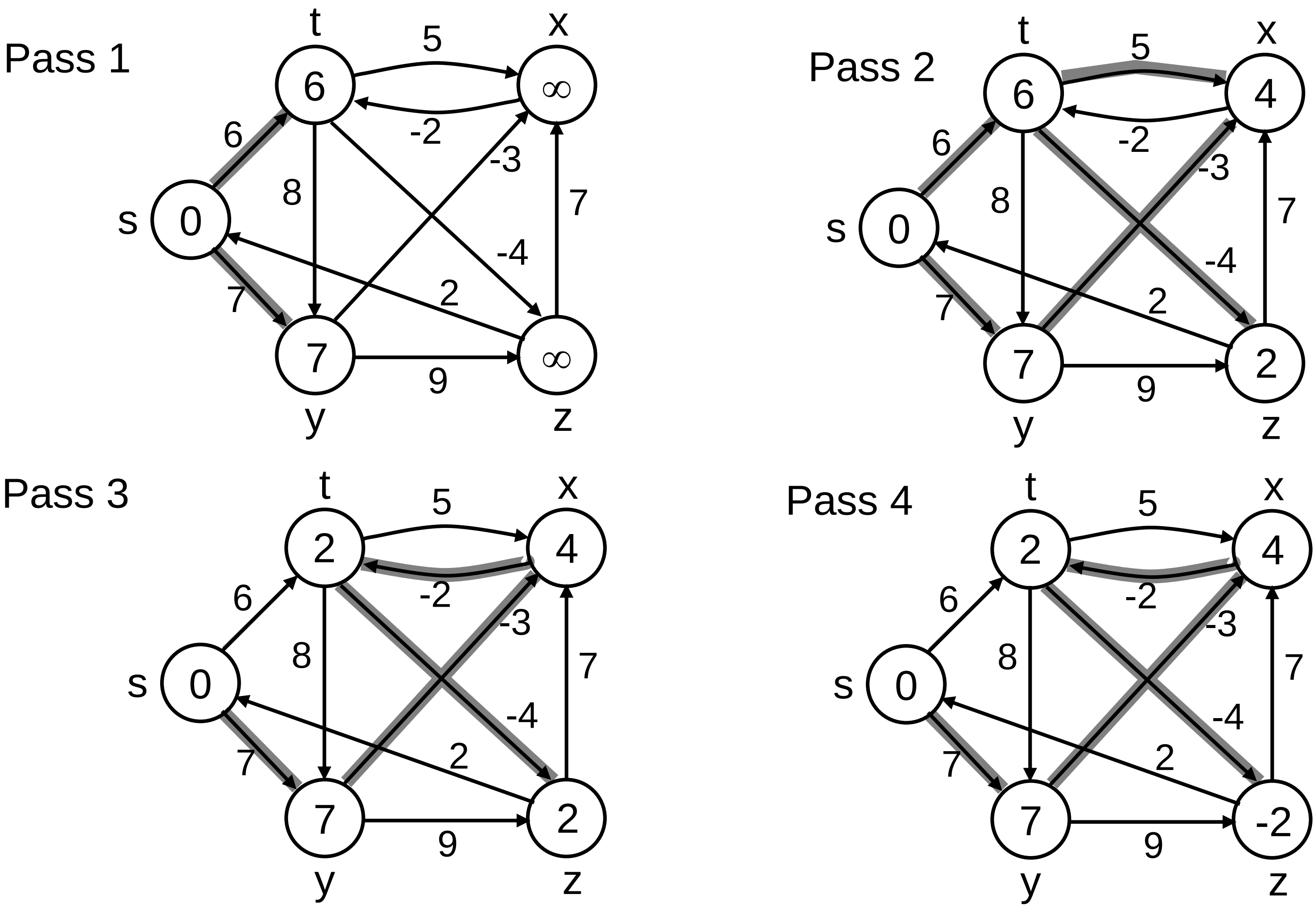
```
INITIALIZE-SINGLE-SOURCE(V, s)
for i ← 1 to |V| - 1
    do for each edge (u, v) ∈ E
            do RELAX(u, v, w)
for each edge (u, v) ∈ E
    do if d[v] > d[u] + w(u, v)
          then return FALSE
return TRUE
```

Example



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

Example



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

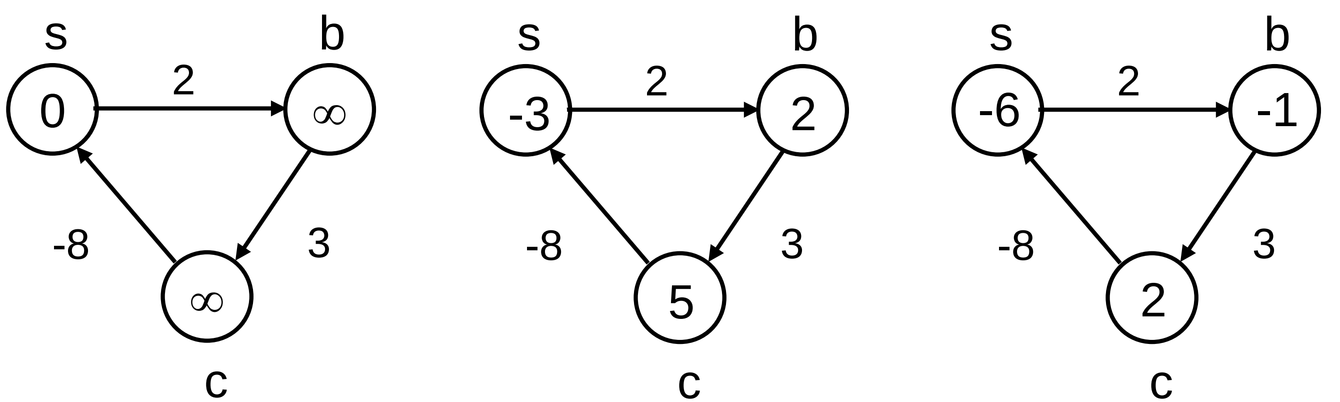
Detecting Negative Cycles

for each edge $(u, v) \in E$

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

then return FALSE

return TRUE



Observe edge (s, b):

$d[b] = -1$, $d[s] + w(s, b) = -4$

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

BELLMAN-FORD(V, E, w, s)

1. INITIALIZE-SINGLE-SOURCE(V, s) $\longleftarrow \Theta(V)$
2. **for** i ← 1 to |V| - 1 $\longleftarrow O(V)$
3. **do for** each edge (u, v) ∈ E $\longleftarrow O(E)$ O(VE)
4. **do** RELAX(u, v, w)
5. **for** each edge (u, v) ∈ E $\longleftarrow O(E)$
6. **do if** d[v] > d[u] + w(u, v)
7. **then return** FALSE
8. **return** TRUE

Running time: O(VE)

Single-Source Shortest Paths in DAGs

Given a weighted DAG: G = (V, E) – solve the shortest path problem

Idea:

- Topologically sort the vertices of the graph
- Relax the edges according to the order given by the topological sort
  - for each vertex, we relax each edge that starts from that vertex

Are shortest-paths well defined in a DAG?

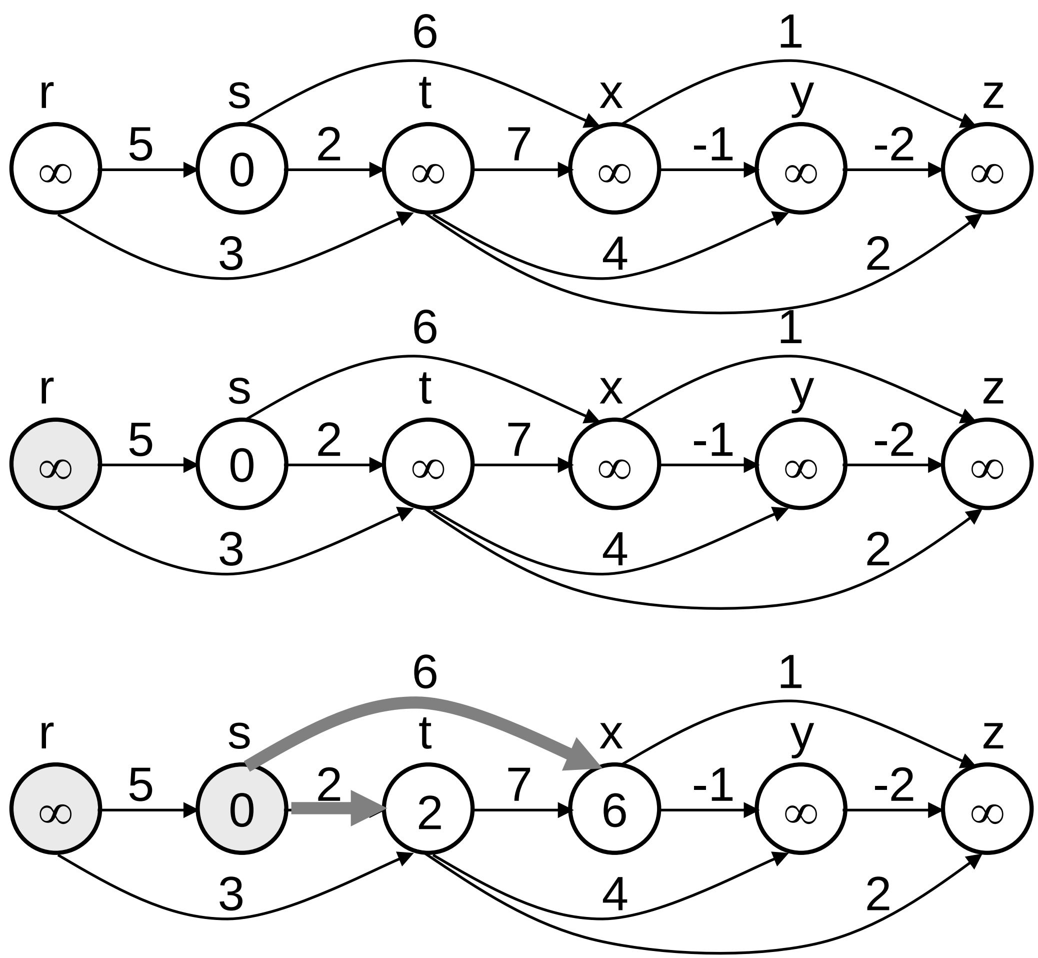
- Yes, (negative-weight) cycles cannot exist

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

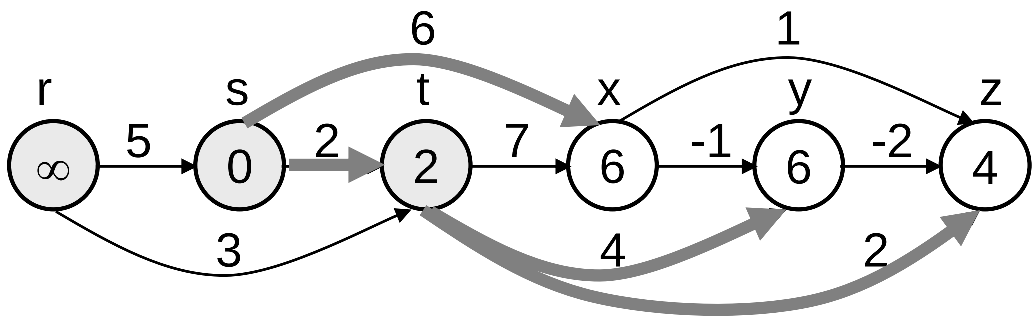
```
topologically sort the vertices of G                          ← Θ(V+E)
INITIALIZE-SINGLE-SOURCE(V, s)                                ← Θ(V)
for each vertex u, taken in topologically sorted order        ← Θ(V)  ⎫
    do for each vertex v ∈ Adj[u]                             ← Θ(E)  ⎬ Θ(V+E)
        do RELAX(u, v, w)                                             ⎭
```

Running time: $\Theta(V+E)$

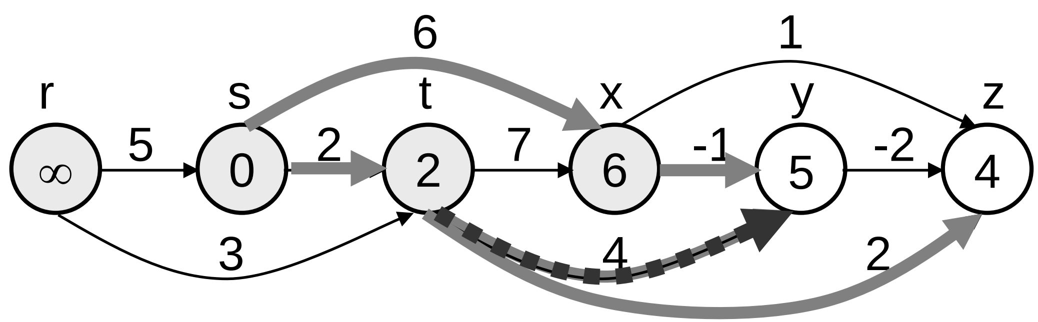
Example



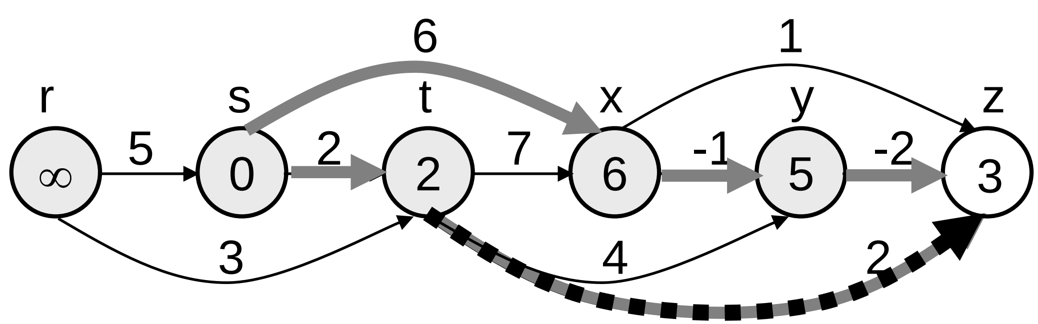
Example



r
s
t
x
y
z
∞
0
2
6
6
4
5
2
7
-1
-2
6
1
3
4
2



r
s
t
x
y
z
∞
0
2
6
5
4
5
2
7
-1
-2
6
1
3
4
2



r
s
t
x
y
z
∞
0
2
6
5
3
5
2
7
-1
-2
6
1
3
4
2

Example