Single-Source Shortest Paths

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:

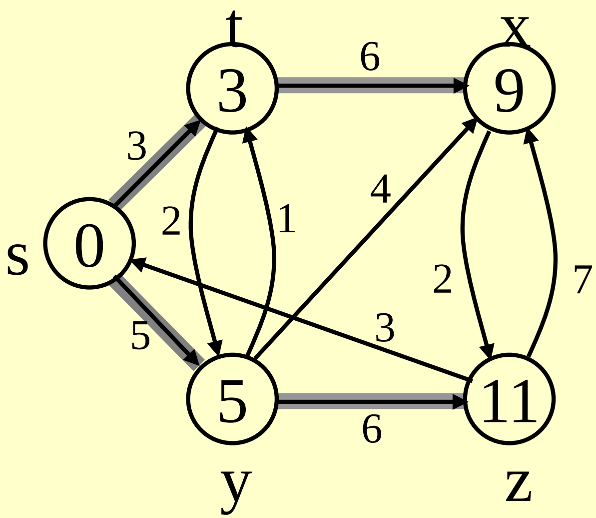
    vertices = cities

    edges = road segments between cities

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

Shortest Path Problem

- **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:

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

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

Variants of Shortest Path

- **Single-source shortest paths**
  - G = (V, E) ⇒ find a shortest path from a given source vertex ***s*** to each vertex **v** ∈ **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 ⇒ single-source

Variants of Shortest Paths (cont'd)

- **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

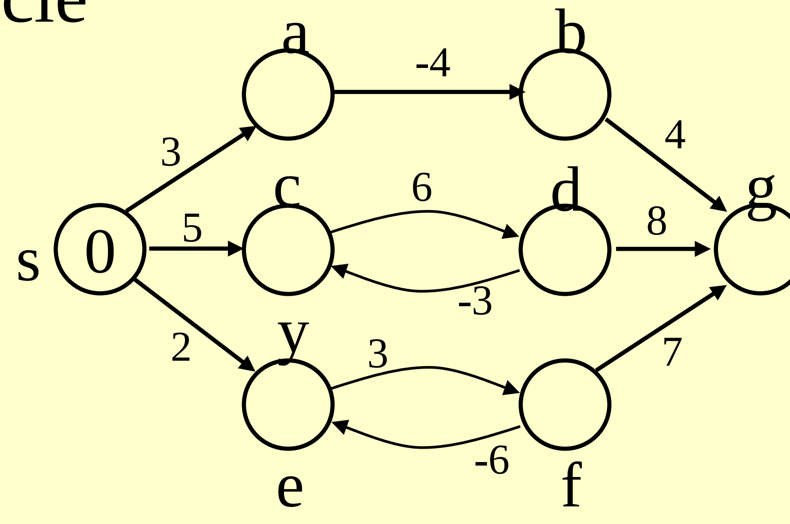
Single-Source Shortest Path Problem

- **Given:** A single source vertex in a weighted, directed graph.
- Want to compute a shortest path for each possible destination.
  - Similar to BFS.
- We will assume either
  - no negative-weight edges, or
  - no reachable negative-weight cycles.
- Algorithm will compute a shortest-path tree.
  - Similar to BFS tree.

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) = - ∞ for all v on the cycle



Negative-Weight Edges

- s → a: only one path

  δ(s, a) = w(s, a) = 3

- s → b: only one path

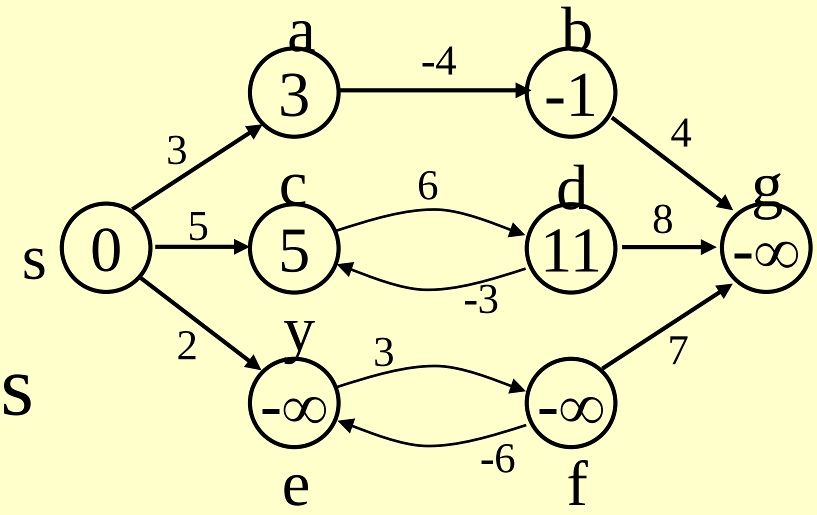
  δ(s, b) = w(s, a) + w(a, b) = -1

- s → c: infinitely many paths

  ⟨s, c⟩, ⟨s, c, d, c⟩, ⟨s, c, d, c, d, c⟩

  cycle has positive weight (6 - 3 = 3)

  ⟨s, c⟩ is shortest path with weight δ(s, b) = w(s, c) = 5

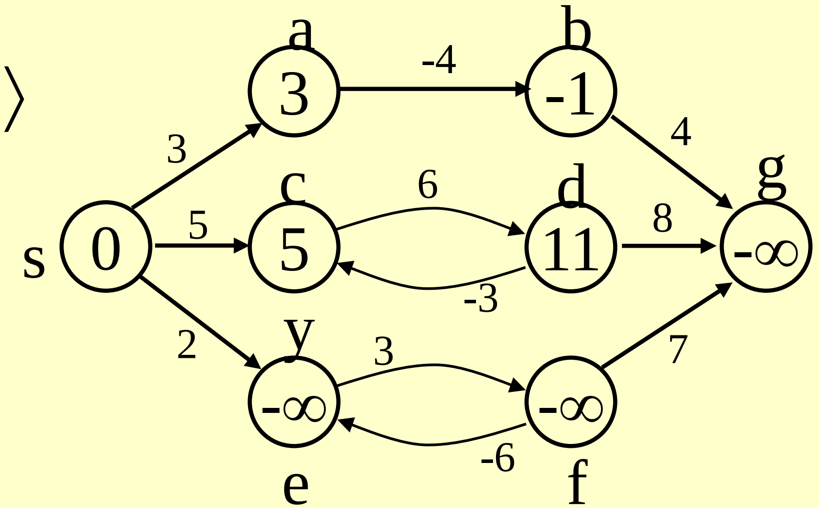


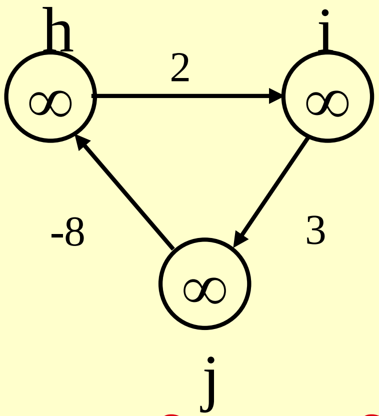
Negative-Weight Edges

- $s \to e$: 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: $\delta(s, f) = -\infty$,

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





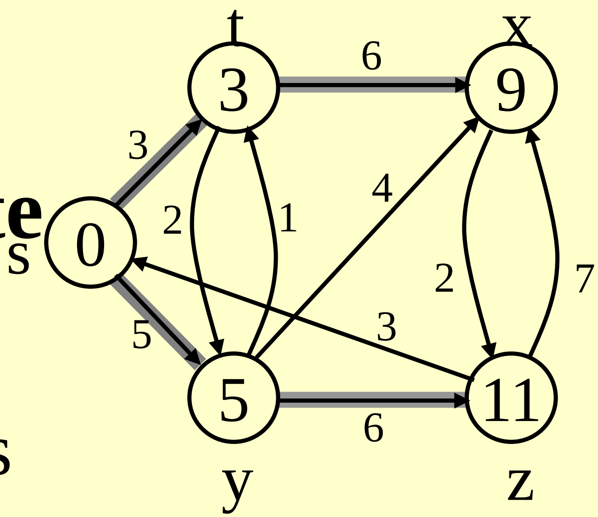
h, i, j not reachable from s

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

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

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

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

1. **for** each $v \in V$

2. **do** $d[v] := \infty$

3. $\pi[v] :=$ NIL

4. $d[s] := 0$

Relaxation Step

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

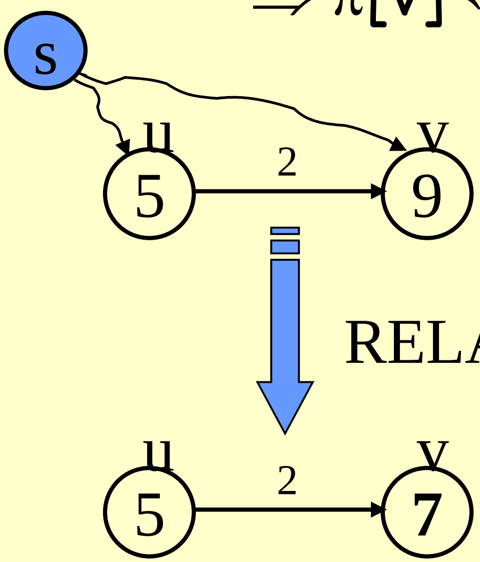
  If $d[v] > d[u] + w(u, v)$

  we can improve the shortest path to v

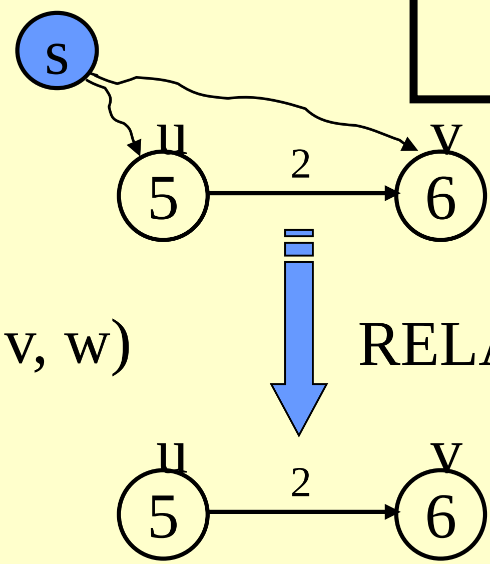
  $\Rightarrow d[v]=d[u]+w(u,v)$

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

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



RELAX(u, v, w)



RELAX(u, v, w)

no change

Relaxation

- Algorithms keep track of d[v], π[v]. These values are changed when an edge (u, v) is **relaxed**:

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

Dijkstra’s Algorithm

- Assumes **no negative-weight edges**.
- Maintains a set S of vertices whose SP from s has been determined.
- Repeatedly selects u in V–S with minimum SP estimate (greedy choice).
- Store V–S in priority queue Q.

Dijkstra's Algorithm

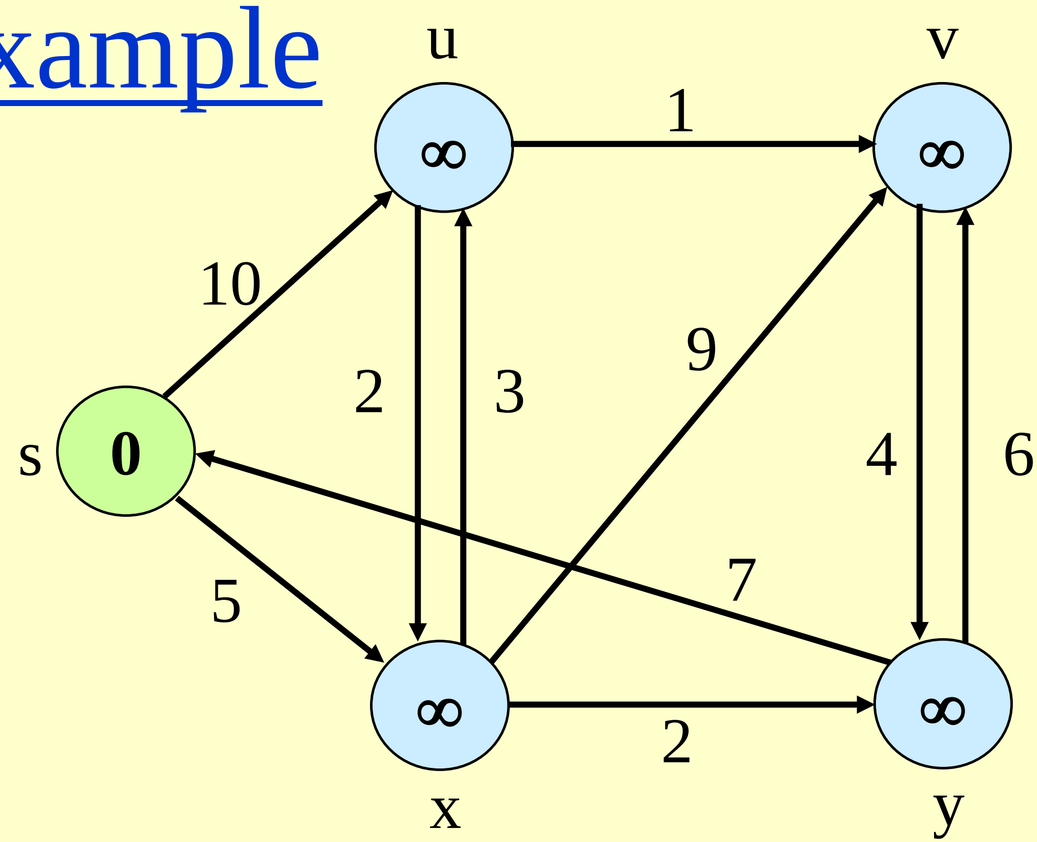
INITIALIZE(G, s);
S := $\emptyset$;
Q := V[G];
while Q $\neq \emptyset$ **do**
    u := Extract-Min(Q);
    S := S $\cup$ {u};
    for each v $\in$ Adj[u] **do**
        Relax(u, v, w)
    od
od

Alg.: INITIALIZE(G, s)

1. **for** each v $\in$ V
2. **do** d[v] := ∞
3. π[v] := NIL
4. d[s] := 0

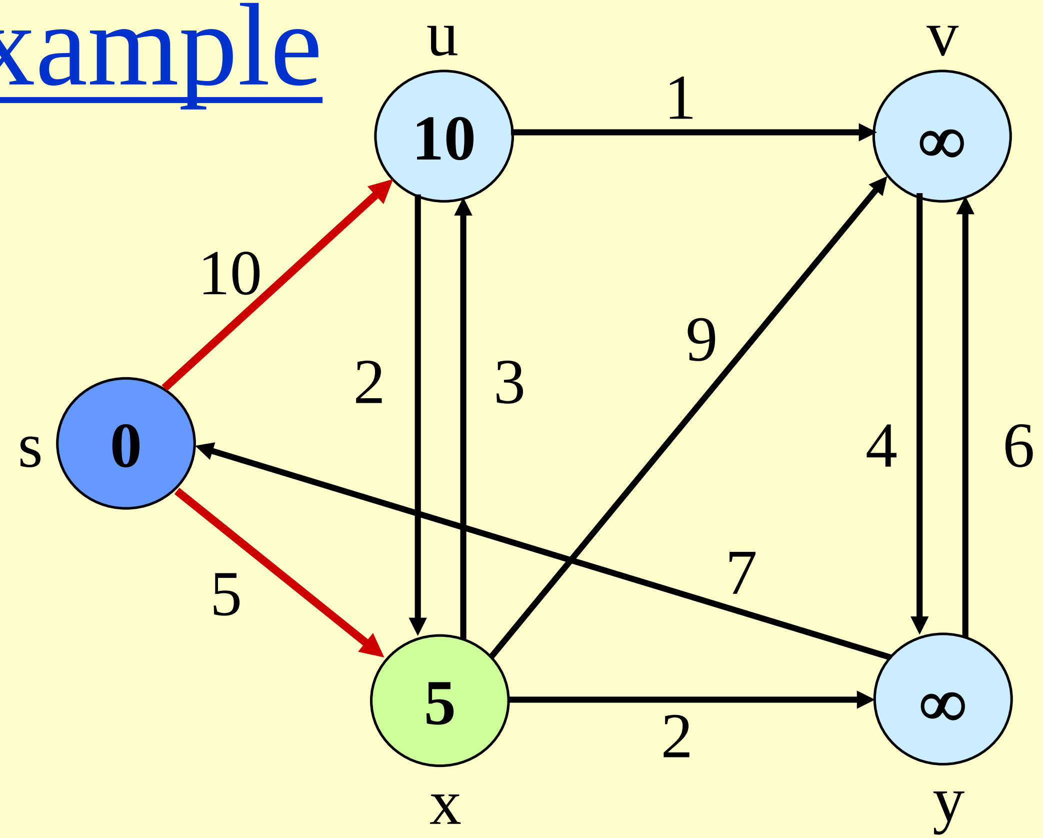
Relax(u, v, w)
    if d[v] > d[u] + w(u, v) **then**
        d[v] := d[u] + w(u, v);
        π[v] := u
    fi

Example



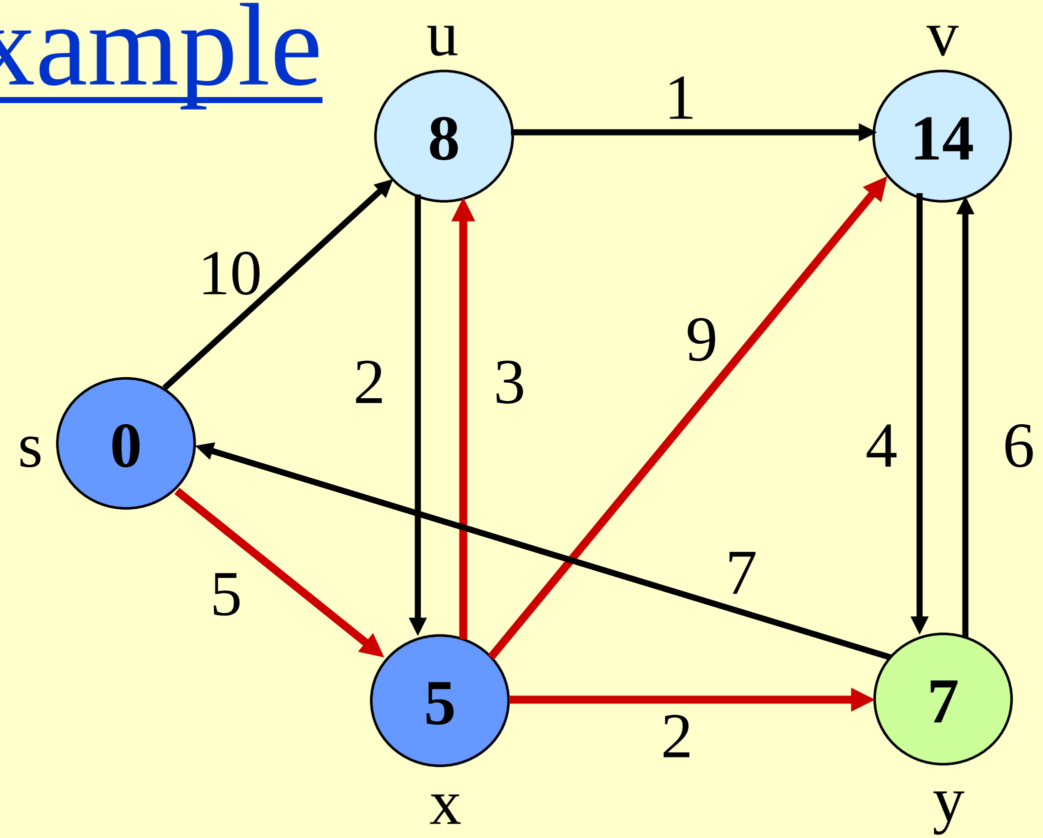
Steps	s	u.d, u.π	v.d, v.π	x.d, x.π	y.d, y.π
0	-	∞, -	∞, -	∞, -	∞, -

Example



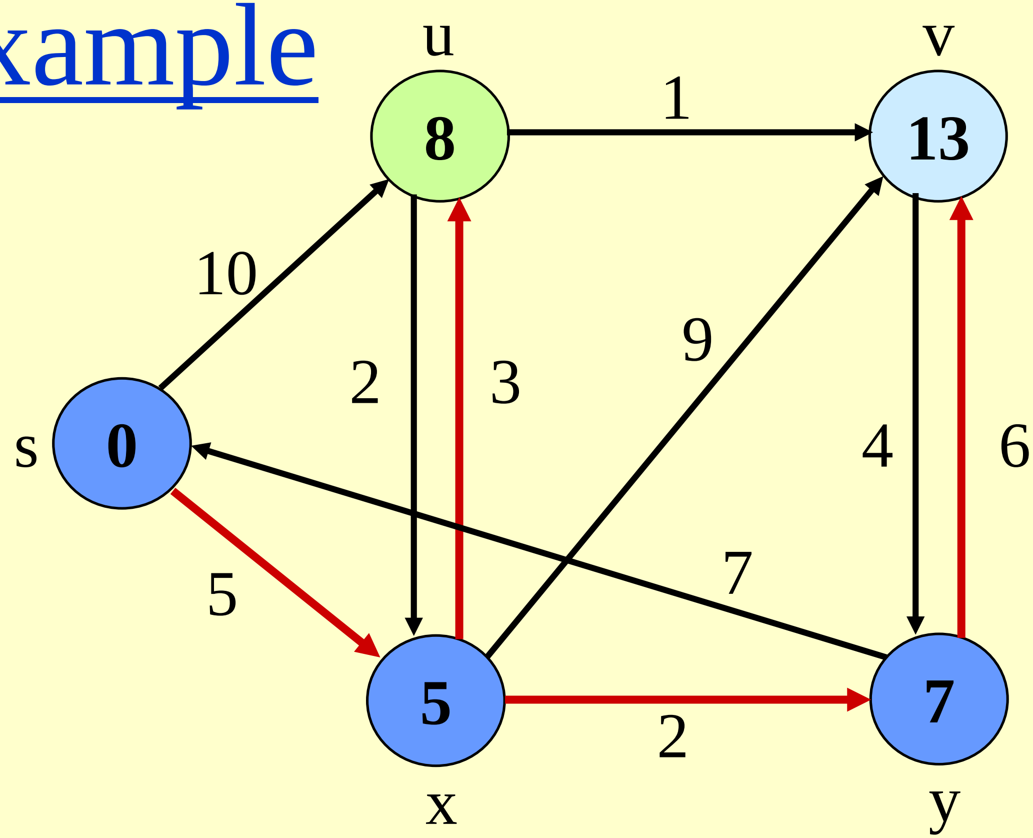
Steps	S	u.d, u.π	v.d, v.π	x.d, x.π	y.d, y.π
0	-	∞, -	∞, -	∞, -	∞, -
1	s	10, s	∞, -	5, s	∞, -

Example



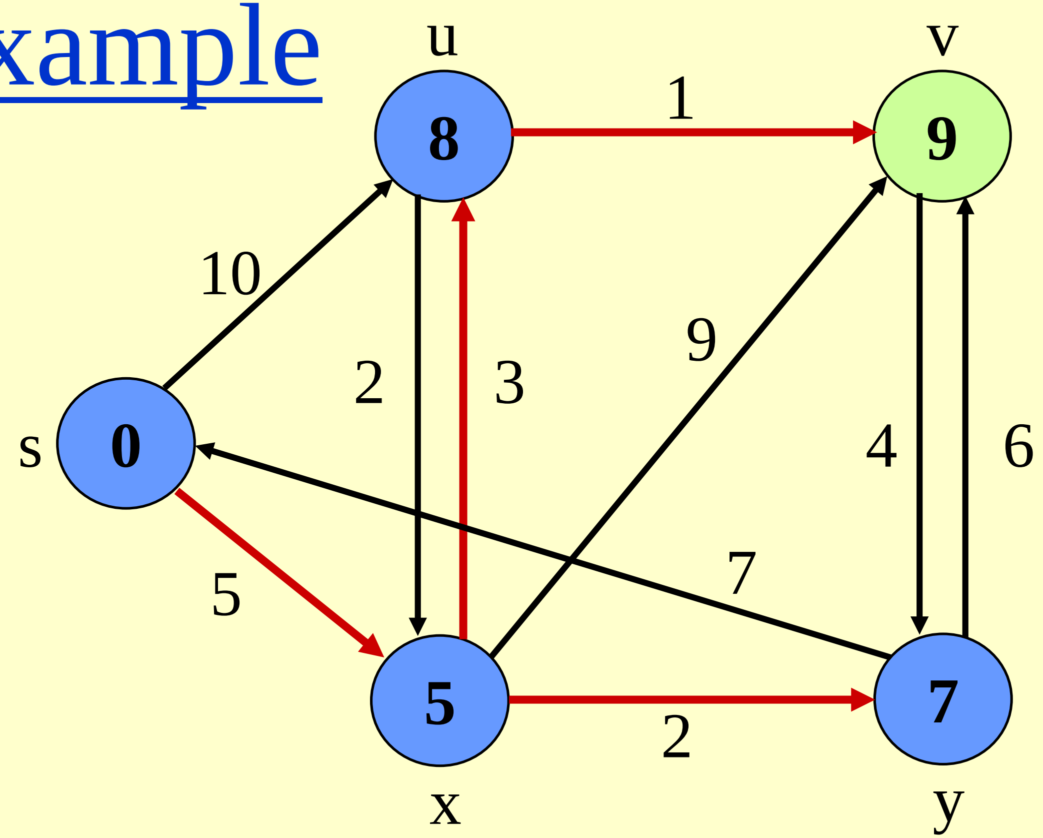
Steps	S	u.d, u.π	v.d, v.π	x.d, x.π	y.d, y.π
0	-	∞, -	∞, -	∞, -	∞, -
1	s	10, s	∞, -	5, s	∞, -
2	s, x	8, x	14, x		7, x

Example



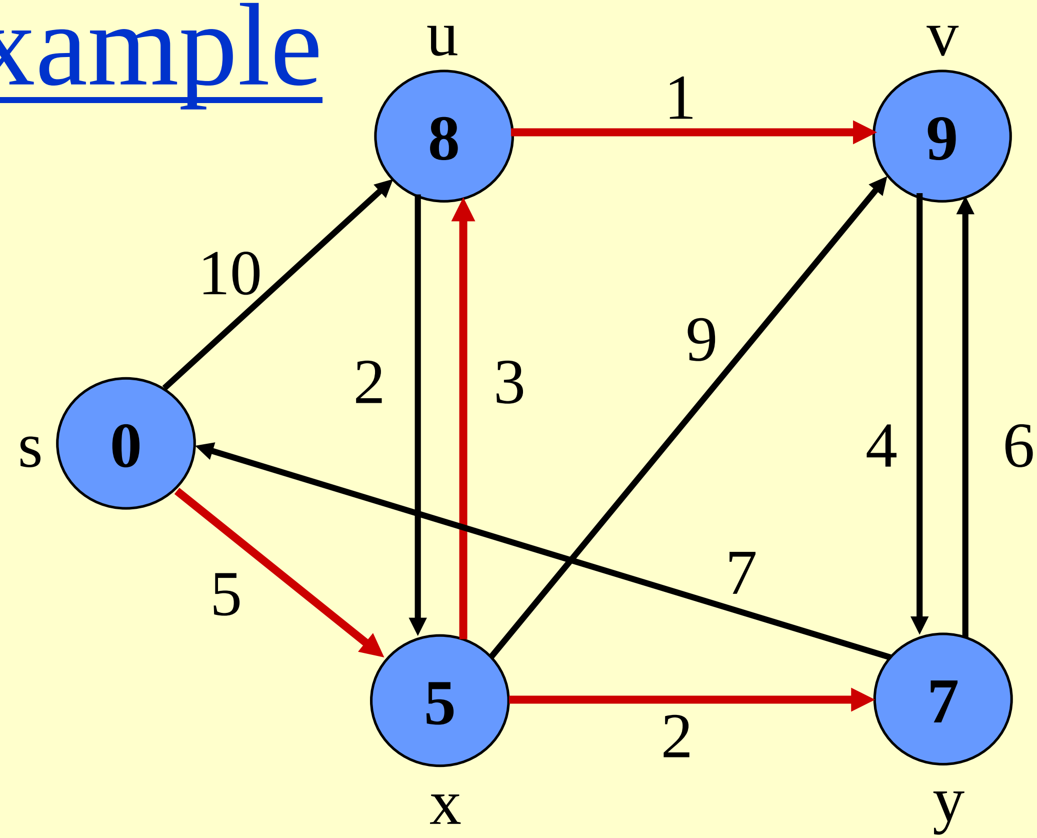
Steps	S	u.d, u.π	v.d, v.π	x.d, x.π	y.d, y.π
0	-	∞, -	∞, -	∞, -	∞, -
1	s	10, s	∞, -	5, s	∞, -
2	sx	8, x	14, x		7, x
3	sxy	8, x	13, y		

Example



Steps	S	u.d, u.π	v.d, v.π	x.d, x.π	y.d, y.π
0	-	∞, -	∞, -	∞, -	∞, -
1	s	10, s	∞, -	5, s	∞, -
2	sx	8, x	14, x		7, x
3	sxy	8, x	13, x		
4	sxyu		9, u		

Example



Steps	S	u.d, u.π	v.d, v.π	x.d, x.π	y.d, y.π
0	-	∞, -	∞, -	∞, -	∞, -
1	s	10, s	∞, -	5, s	∞, -
2	sx	8, x	14, x		7, x
3	sxy	8, x	13, x		
4	sxyu		9, u		
5	sxyuv				

Complexity of Dijkstra (G, w, s)

INITIALIZE-SINGLE-SOURCE(V, s) $\longleftarrow$ $\Theta(V)$

S $\leftarrow \emptyset$

Q $\leftarrow$ V[G] $\longleftarrow$ O(V) build min-heap

while Q $\neq \emptyset$ $\longleftarrow$ Executed O(V) times

  do u $\leftarrow$ EXTRACT-MIN(Q) $\longleftarrow$ O(lgV)

O(VlgV)

    S $\leftarrow$ S $\cup$ {u}

    for each vertex v $\in$ Adj[u] $\longleftarrow$ O(V+E) times (total)

      do RELAX(u, v, w)

      Update Q (DECREASE_KEY) $\longleftarrow$ O(lgV)

O((V+E)lg

Running time: $O(VlgV + (V+E)lgV) = O((V+E)lgV)$

Complexity

- Running time is
  - $O(V^2)$ using linear array for priority queue.
  - $O((V + E) \lg V)$ using binary heap.