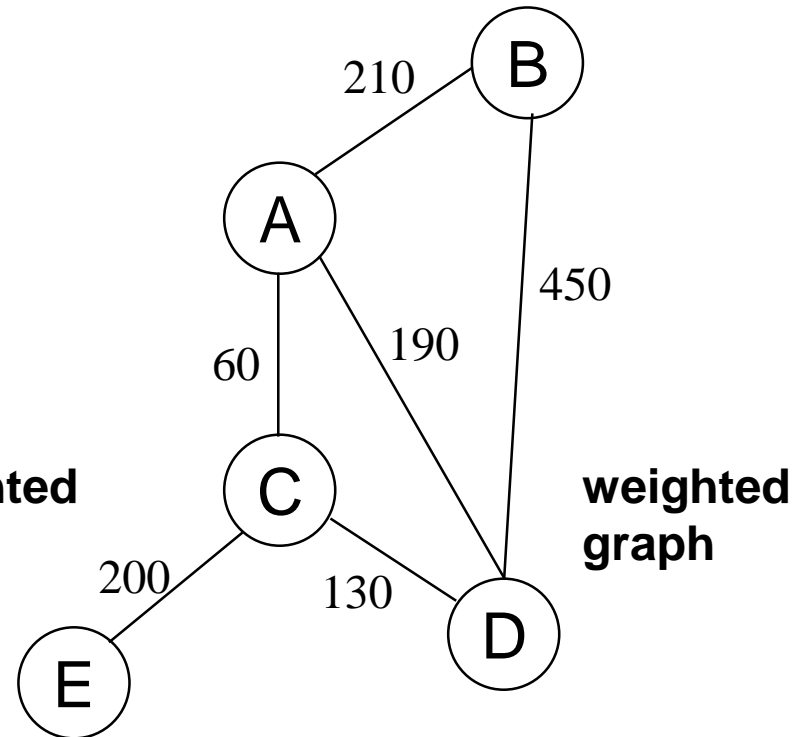
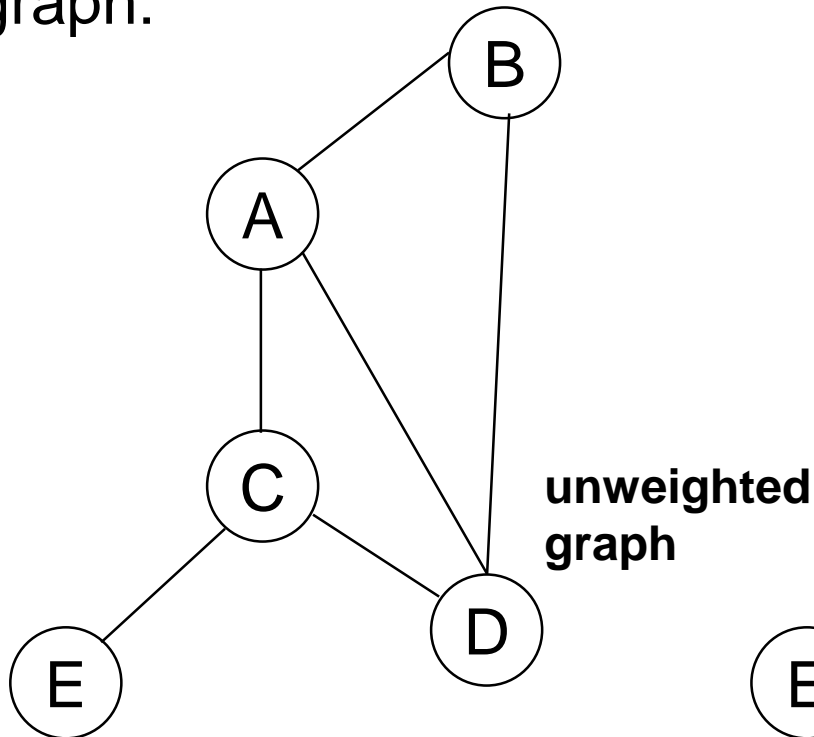


Design & Analysis of Algorithm

Single Source Shortest Path
(Dijkstra's Algorithm)

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

- How can we find the shortest route between two points on a 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 Problems

- **Input:**

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

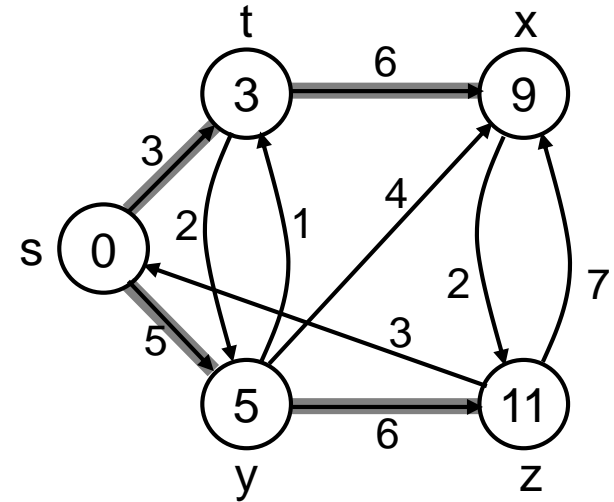
- **Weight of path $p = \langle v_0, v_1, \dots, v_k \rangle$**

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

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

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

- Shortest path u to v is any path p such that $w(p) = \delta(u, v)$



Variants of Shortest Paths

- **Single-source shortest path**

- $G = (V, E) \Rightarrow$ 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

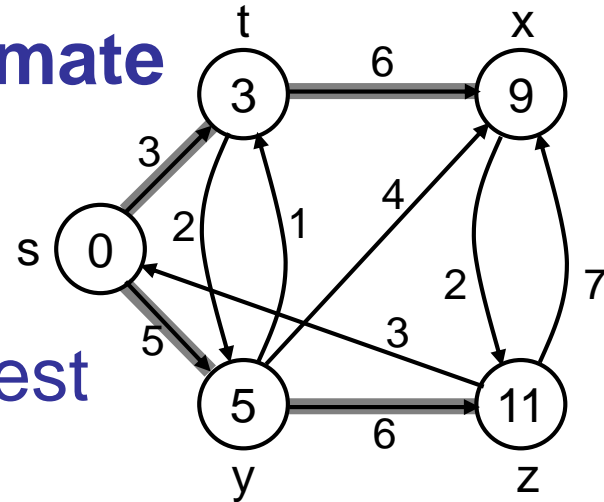
- **All-pairs shortest-paths**

- 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, $\pi[v] = \text{NIL}$
 - 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. $\pi[v] \leftarrow \text{NIL}$
 4. $d[s] \leftarrow 0$
- All the shortest-paths algorithms start with INITIALIZE-SINGLE-SOURCE

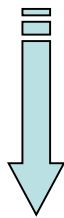
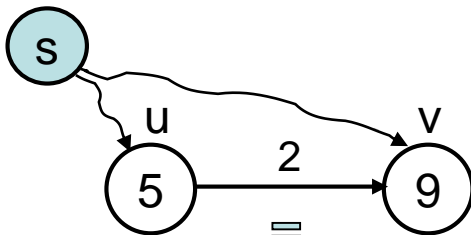
Relaxation

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

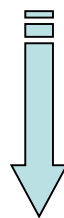
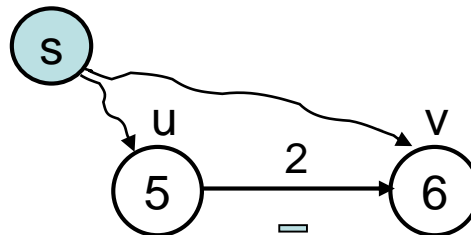
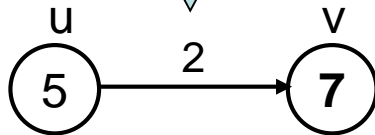
If $d[v] > d[u] + w(u, v)$

we can improve the shortest path to v

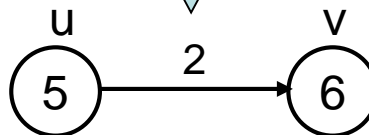
\Rightarrow update $d[v]$ and $\pi[v]$



RELAX(u, v, w)



RELAX(u, v, w)



After relaxation:

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

RELAX(u, v, w)

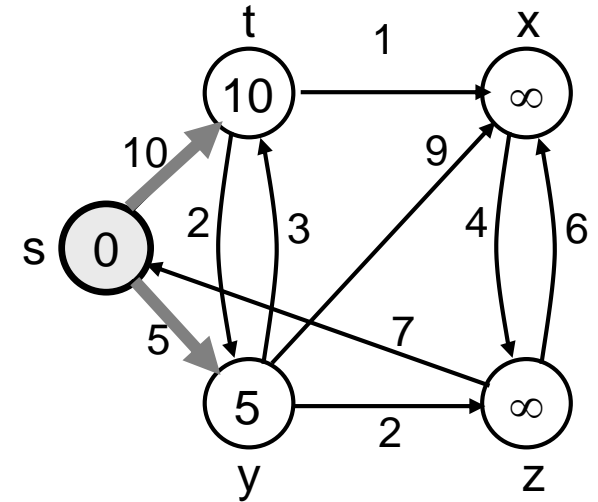
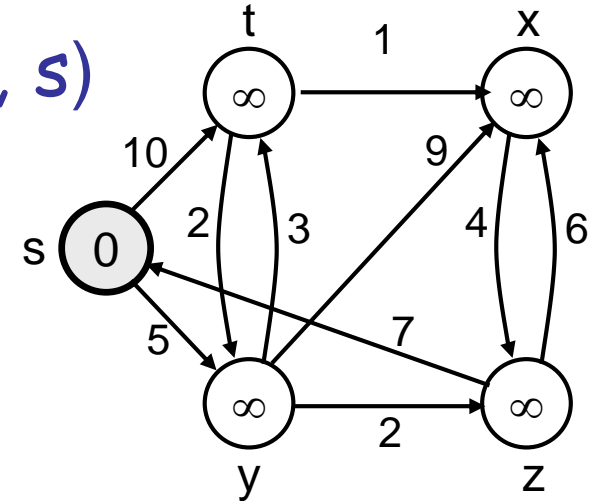
1. **if** $d[v] > d[u] + w(u, v)$
 2. **then** $d[v] \leftarrow d[u] + w(u, v)$
 3. $\pi[v] \leftarrow u$
-
- All the single-source shortest-paths algorithms
 - start by calling INIT-SINGLE-SOURCE
 - then relax edges
 - The algorithms differ in the order and how many times they relax each edge

Dijkstra's Algorithm

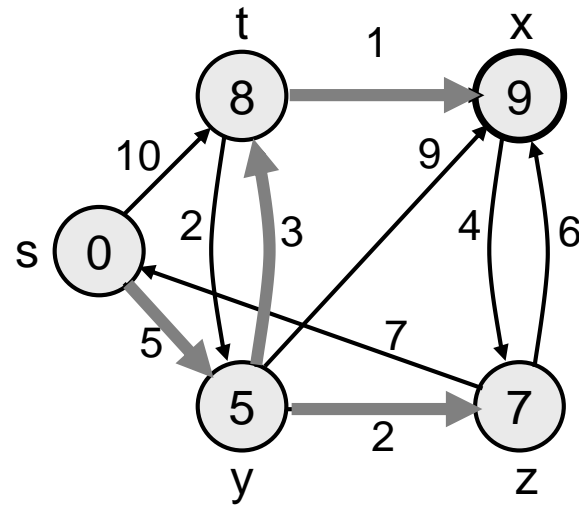
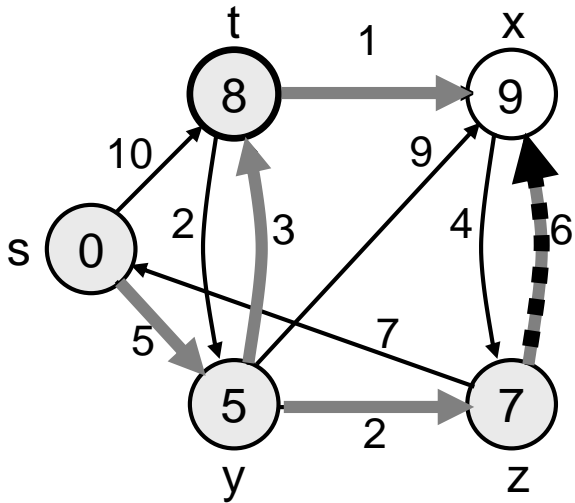
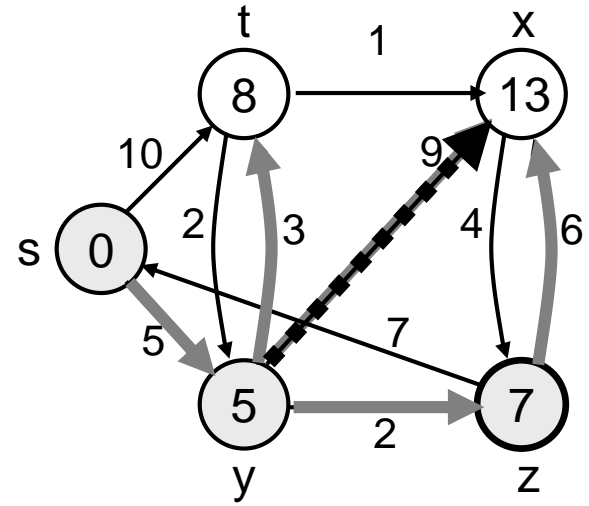
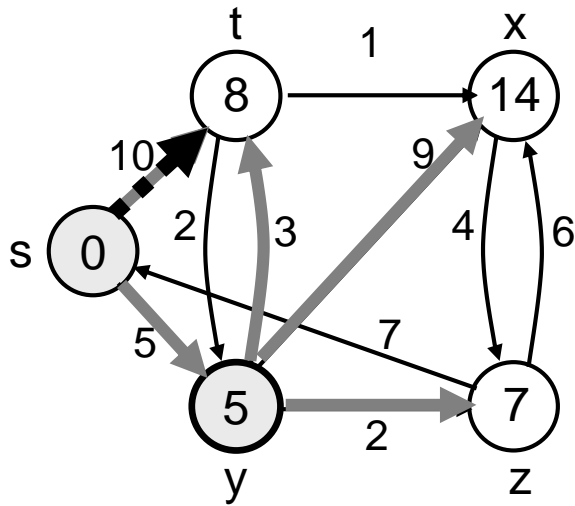
- Single-source shortest path problem:
 - No negative-weight edges: $w(u, v) > 0 \forall (u, v) \in E$
- Maintains two sets of vertices:
 - S = vertices whose final shortest-path weights have already been determined
 - Q = vertices in $V - S$: min-priority queue
 - Keys in Q are estimates of shortest-path weights ($d[v]$)
- Repeatedly select a vertex $u \in V - S$, with the minimum shortest-path estimate $d[v]$

Dijkstra (G, w, s)

1. INITIALIZE-SINGLE-SOURCE(V, s)
2. $S \leftarrow \emptyset$
3. $Q \leftarrow V[G]$
4. **while** $Q \neq \emptyset$
5. **do** $u \leftarrow \text{EXTRACT-MIN}(Q)$
6. $S \leftarrow S \cup \{u\}$
7. **for** each vertex $v \in \text{Adj}[u]$
8. **do** RELAX(u, v, w)



Example



Dijkstra's Pseudo Code

- Graph G , weight function w , root s

DIJKSTRA(G, w, s)

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

2 **do** $d[v] \leftarrow \infty$

3 $d[s] \leftarrow 0$

4 $S \leftarrow \emptyset$ \triangleright Set of discovered nodes

5 $Q \leftarrow V$

6 **while** $Q \neq \emptyset$

7 **do** $u \leftarrow \text{EXTRACT-MIN}(Q)$

8 $S \leftarrow S \cup \{u\}$

9 **for** each $v \in \text{Adj}[u]$

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

11 **then** $d[v] \leftarrow d[u] + w(u, v)$

relaxing
edges

Dijkstra (G, w, s)

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

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

Dijkstra's Running Time

- Extract-Min executed $|V|$ time
- Decrease-Key executed $|E|$ time
- Time = $|V| T_{\text{Extract-Min}} + |E| T_{\text{Decrease-Key}}$
- T depends on different Q implementations

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