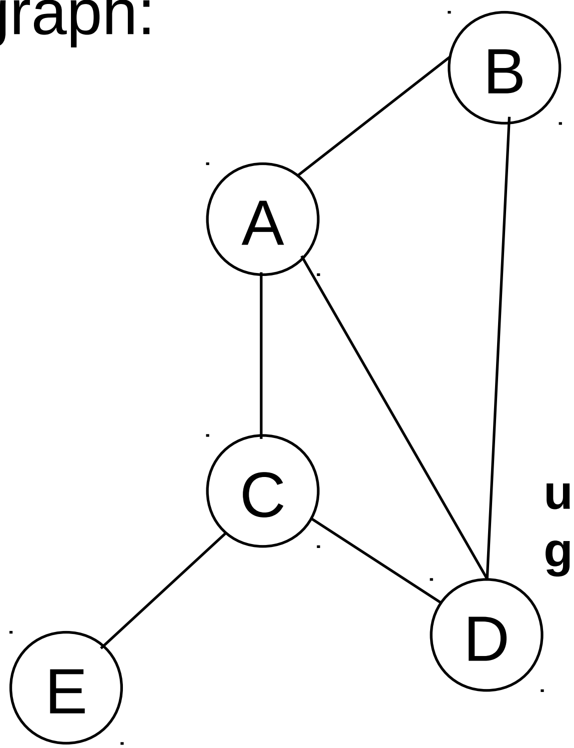
CSE304 – Design & Analysis of Algorithm

Single Source Shortest Path

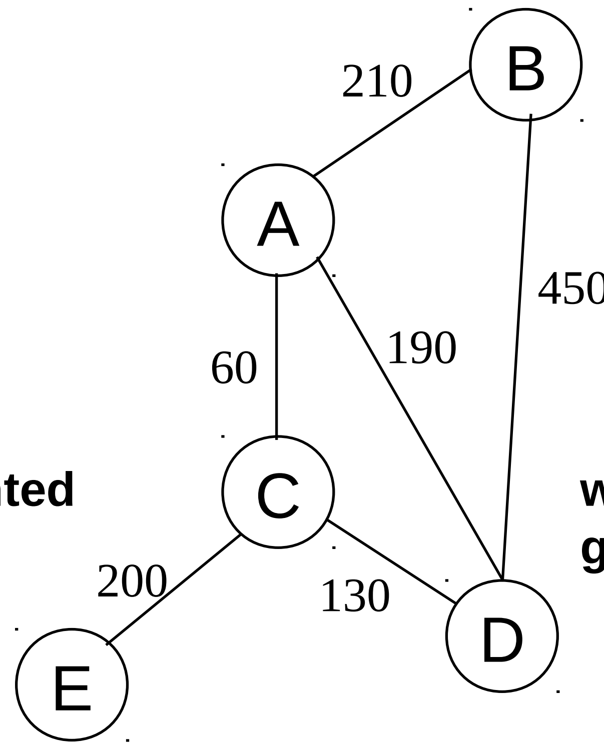
(Dijkstra's Algorithm)

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:



unweighted graph



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:

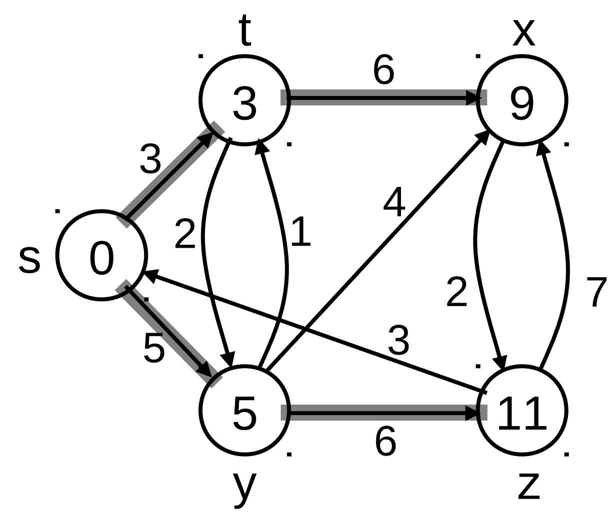
    vertices = cities

    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 → **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:

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

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

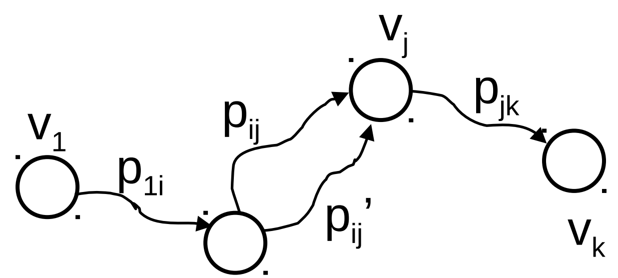
Variants of Shortest Paths

- **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

Optimal Substructure of Shortest Paths

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 $1 \leq i \leq j \leq k$



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

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

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

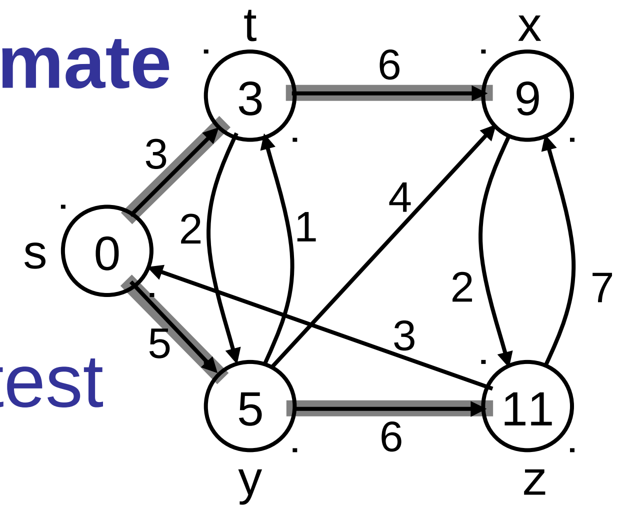
Assume $\exists\, p_{ij}'$ from v_i to v_j with $w(p_{ij}') < w(p_{ij})$

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

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

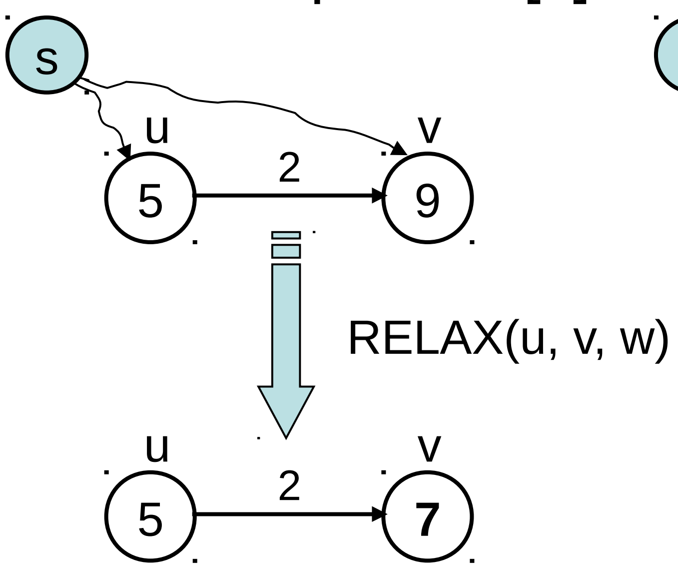
Relaxation

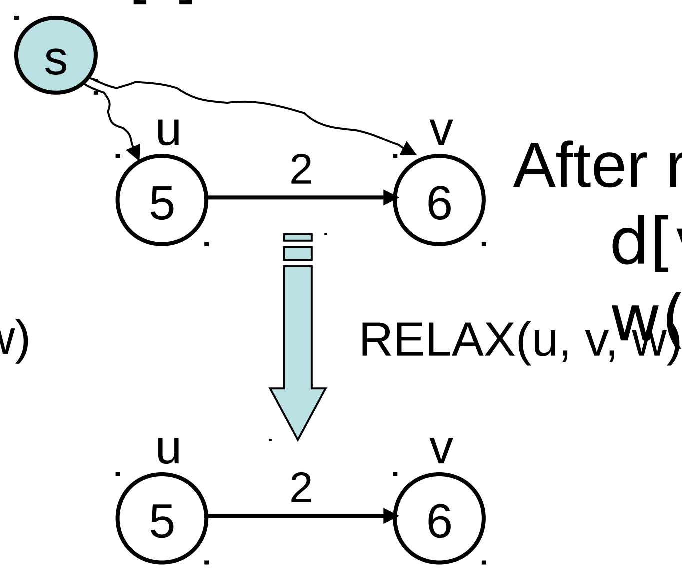
- **Relaxing** an edge (u, v) = testing whether we can improve the shortest path to v found so far by going through u

  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:

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

RELAX(u, v, w)

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

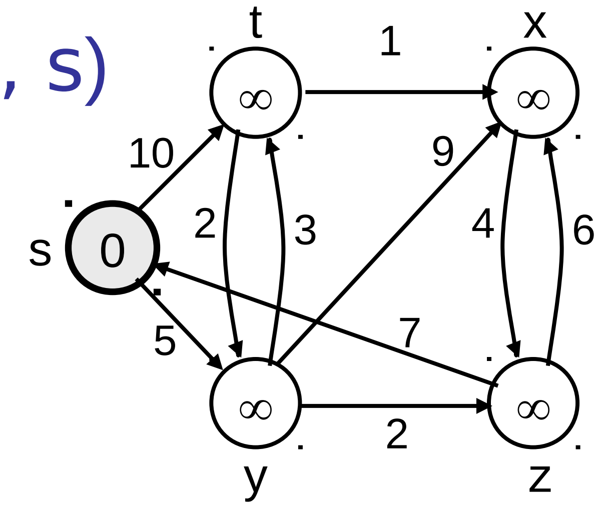
- All the single-source shortest-paths algorithms
  - start by calling INIT-SINGLE-SOURCE
  - then relax edges
- The algorithms differ in the order and how many times they relax each edge

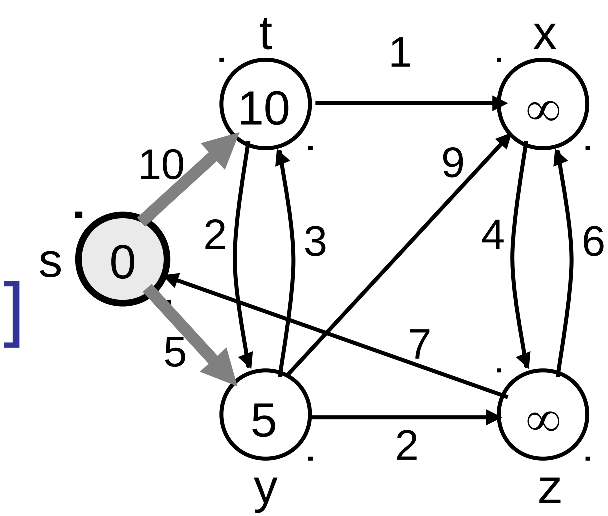
Dijkstra's Algorithm

- 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]

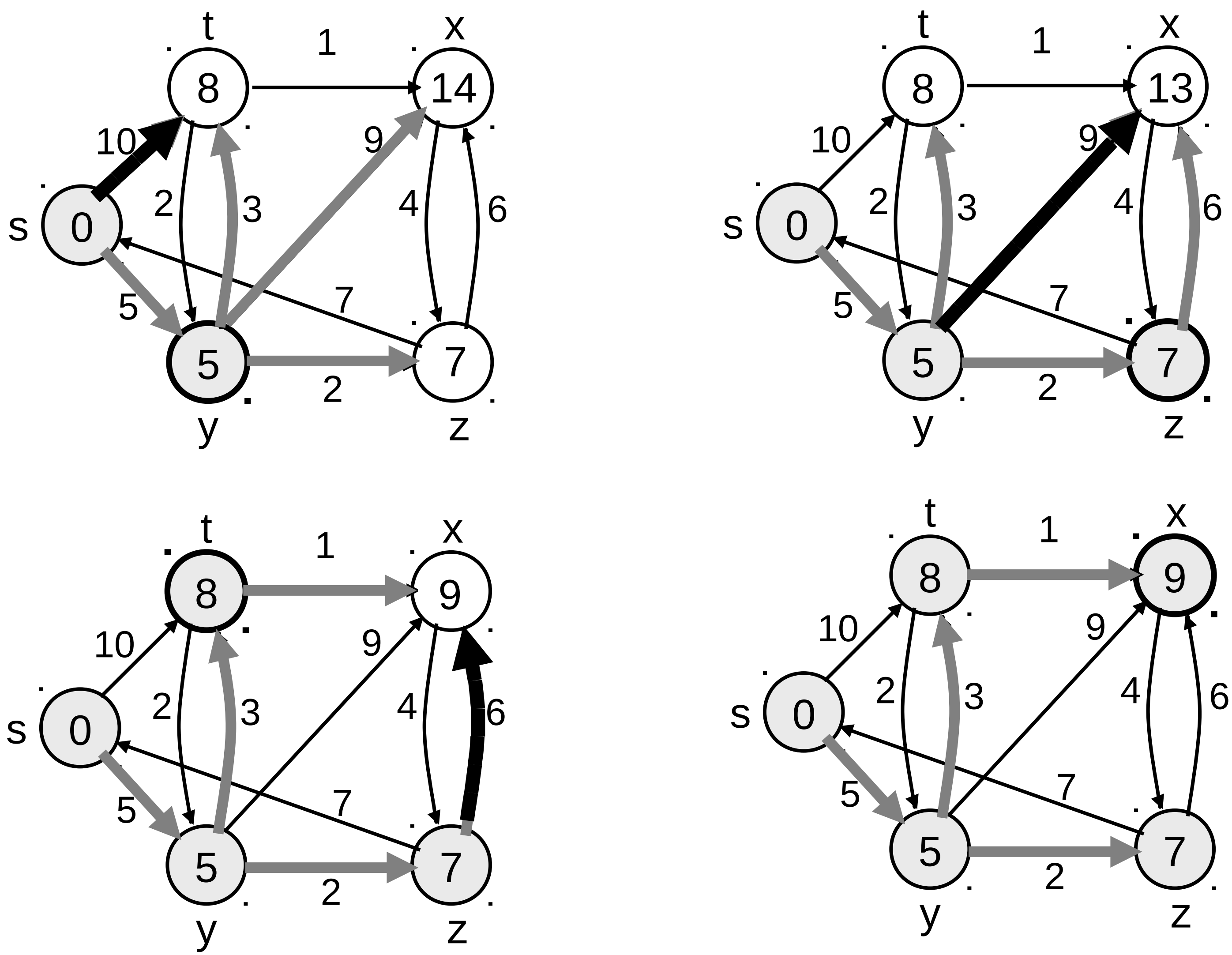
Dijkstra (G, w, s)

1. INITIALIZE-SINGLE-SOURCE(V, s)
2. S ← ∅
3. Q ← V[G]
4. **while** Q ≠ ∅
5.   **do** u ← EXTRACT-MIN(Q)
6.   S ← S ∪ {u}
7.   **for** each vertex v ∈ Adj[u]
8.   **do** RELAX(u, v, w)





Example


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

Dijkstra's Pseudo Code

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

$\textsc{Dijkstra}(G, w, s)$

for each $v \in V$
    do $d[v] \leftarrow \infty$
$d[s] \leftarrow 0$
$S \leftarrow \emptyset$ $\triangleright$ Set of discovered nodes
$Q \leftarrow V$
while $Q \neq \emptyset$
    do $u \leftarrow \textsc{Extract-Min}(Q)$
        $S \leftarrow S \cup \{u\}$
        for each $v \in Adj[u]$
            do if $d[v] > d[u] + w(u, v)$
                then $d[v] \leftarrow d[u] + w(u, v)$

relaxing edges

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)

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

        for each vertex $v \in Adj[u]$

            do RELAX(u, v, w) $\longleftarrow$ O(E) times; O(lgV)

Running time: O(VlgV + ElgV) = O(ElgV)

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.)	$O(V \lg V + E)$

Question

- Prove that, if there exists negative edge, dijkstra's shortest path algorithm may fail to find the shortest path
- Print the shortest path for dijkstra's algorithm
- How many shortest paths are there from source to destination?
- Suppose you are given a graph where each edge represents the path cost and each vertex has also a cost which represents that, if you select a path using this node, the cost will be added with the path cost. How can it be solved using Dijkstra's algorithm?

Question

- How to solve ACM534 – Frogger?