DESIGN AND ANALYSIS OF ALGORITHMS

CS 4120/5120

SHORTEST PATHS ALGORITHM

AGENDA

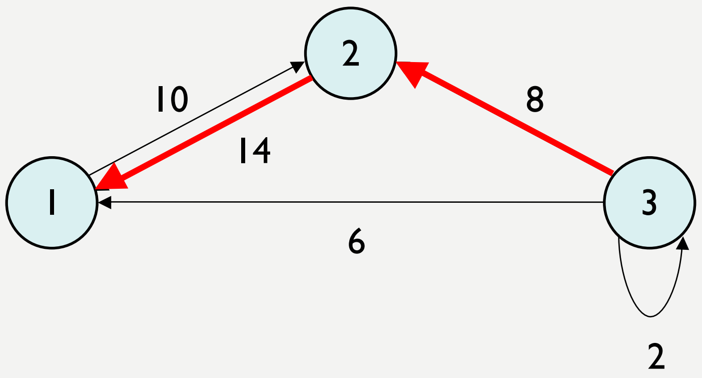
- The definitions
  - The shortest-path problems
  - The optimal substructure
  - The shortest path
- Relaxation technique
- Dijkstra's Algorithm
- Bellman-Ford Algorithm

THE WEIGHT OF A PATH

- Consider a ***weighted directed*** graph $G = (V, E)$ with a ***weight function*** $\boldsymbol{w}: \boldsymbol{E} \to \mathbb{R}$.
- The **weight $w(p)$ of path $\boldsymbol{p}$** $= < \boldsymbol{v_0}, \boldsymbol{v_1}, \dots, \boldsymbol{v_k} >$ is the sum of the weights of its constituent edges:

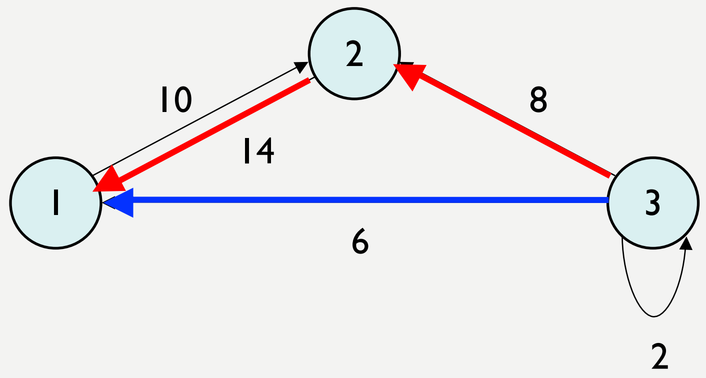
$$w(\boldsymbol{p}) = \sum_{i=1}^{k} w(\boldsymbol{v_{i-1}}, \boldsymbol{v_i})$$

- Example
  - Consider the graph on the right. The weight $\boldsymbol{w}(\boldsymbol{p})$ where $\boldsymbol{p} =< 3,2,1 >$ is $\underline{w(3,2) + w(2,1) = 8 + 14 = 22}$.



THE DEFINITION OF A SHORTEST-PATHS PROBLEM

- There exist an even shorter path from vertex 3 to 1.
  - $w(p') = \underline{\ w(3,1) = 6 < w(\boldsymbol{p})}$



- **Formally**, the ***single-source shortest-paths problem*** is defined as: Given a graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, we want to find a shortest path from a given source vertex $s \in V$ to each vertex $v \in V$.

THE SHORTEST PATH PROBLEMS

- Variants
  - ***Single-destination*** ***shortest-paths problem***
    - Find a shortest path to a given ***destination*** vertex t from each vertex v.
  - ***Single-pair*** ***shortest-paths problem***: Find a shortest path from u to v for given vertices u and v.
  - ***All-pairs shortest-paths*** ***problem***: Find a shortest path from u to v for every pair of vertices u and v.

- All the variants can be solved by solving the ***single-source*** ***shortest-paths (SP) problem***.
  - All-pairs shortest-paths problem have more efficient solution.

THE DEFINITION OF A SHORTEST PATH IN A SP PROBLEM

- We define the ***shortest-path weight*** $\delta(u, v)$ from u to v by

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

  – A path from vertex u to v is denoted by $u \overset{p}{\leadsto} v$.

- A ***shortest path*** from vertex u to vertex v is then defined as any path p with weight $w(p) = \delta(u, v)$.

PERFECTING THE DEFINITION OF A SHORTEST PATH #1 NEGATIVE EDGE

- The ***shortest-path weight*** $\boldsymbol{\delta(u, v)}$ from u to v by

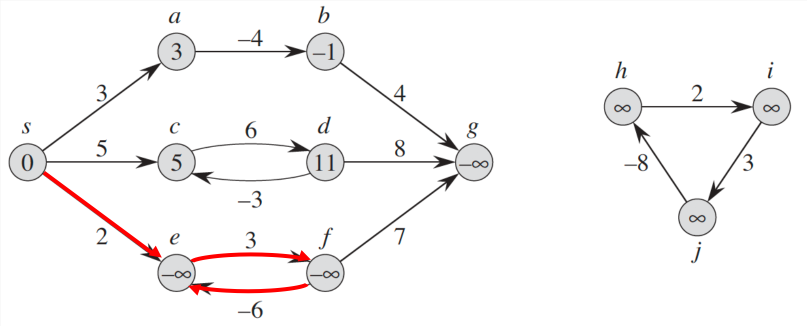
$$\boldsymbol{\delta(u,v)} = \begin{cases} \mathbf{min}\{w(p): \boldsymbol{u} \overset{\boldsymbol{p}}{\leadsto} \boldsymbol{v}\} & \text{if there is a path from } u \text{ to } v, \\ \infty & \text{otherwise.} \end{cases}$$

  – Works well **if there is NO negative weighted edge**.

- Example of a graph with negative weighted edges

  – In this case, we can always best the proposed "shortest path" by including **a negative-weight cycle** in the path.

- If there is a **negative weight cycle** on some path from s to v, we define $\boldsymbol{\delta(s, v) = -\infty}$.

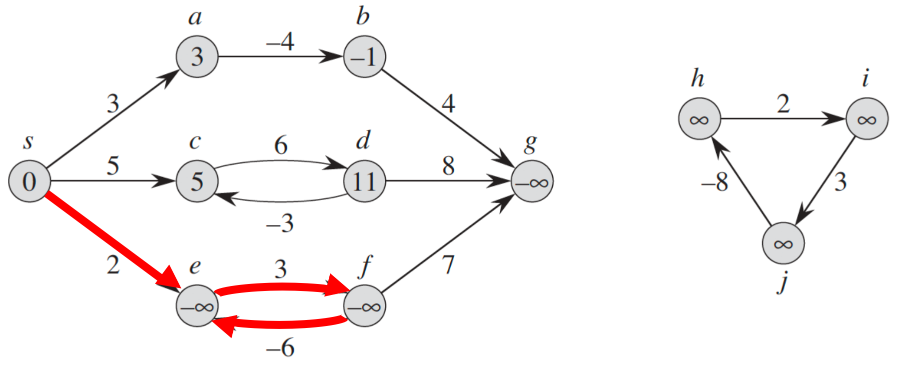


PERFECTING THE DEFINITION OF A SHORTEST PATH #2 NO CYCLE

- The ***shortest-path weight*** $\delta(u, v)$ from u to v by

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

  - Works well **if there is NO cycle in the graph**.
- Certainly, a shortest path CANNOT contain a cycle
  - Neither a negative-weight cycle NOR
  - a positive-weight cycle
    - We can always obtain a path with a lower weight by removing the cycle from the proposed "shortest path"



THE DEFINITION OF A SHORTEST PATH FINAL

- We define the ***shortest-path weight*** $\boldsymbol{\delta(u,v)}$ from u to v by

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

, where the path $\boldsymbol{u} \overset{\boldsymbol{p}}{\leadsto} \boldsymbol{v}$ contains NO negative-weighted edge or cycle.

  – If If there is a **negative weight cycle** on some path from s to v, we define $\boldsymbol{\delta(s,v)} = -\infty$.

THE OPTIMAL SUBSTRUCTURE OF SHORTEST PATH LEMMA 24.1

- Given a **weighted**, **directed** graph $G = (V, E)$ with **weight function** $\boldsymbol{w}: E \to \mathbb{R}$.
- Let $\boldsymbol{p} = < \boldsymbol{v_0}, v_1, \dots, \boldsymbol{v_k} >$ be a ***shortest*** path from vertex $\boldsymbol{v_0}$ to vertex $\boldsymbol{v_k}$, and for any i and j such that $0 \le i \le j \le k$, let $p_{ij} = < v_i, v_{i+1}, \dots, v_j >$ be the ***subpath*** of $\boldsymbol{p}$ from vertex v_i to vertex v_j.
- Then, $\boldsymbol{p_{ij}}$ **is a shortest path from $\boldsymbol{i}$ to $\boldsymbol{j}$**.
- In other words, the **subpaths of shortest paths are shortest paths**.

THE OPTIMAL SUBSTRUCTURE OF SHORTEST PATH LEMMA 24.1 PROOF

- Use "cut-and-paste" technique.
- The goal is to prove that $p_{ij} = < v_i, v_{i+1}, \dots, v_j >$, where $0 \le i \le j \le k$, contained within a shortest path $\boldsymbol{p} = < \boldsymbol{v_0}, v_1, \dots, \boldsymbol{v_k} >$ is a shortest path from v_i to v_j.

i. **Assume** ____.

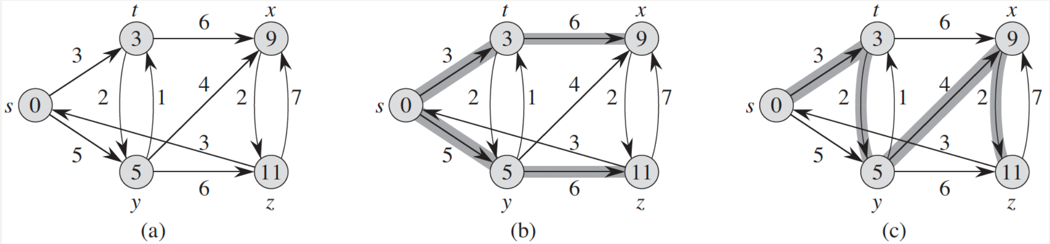
ii. There exist a shortest path, denoted by p'_{ij}, from _____ to _____, obviously, _________ $\le$ _________.

iii. We can **construct a new path**, denoted by $\boldsymbol{p}^*$, by cutting _____ out of _____ then pasting _____.

iv. As a result, , _________ $\le$ _________, **contradicting the supposition** that _____ is a shortest path from vertex _____ to _____.

REPRESENTING A SHORTEST PATH
SHORTEST PATH TREE

- To be precise, let $G = (V, E)$ be a ***weighted***, ***directed*** graph with **weight function $\boldsymbol{w}$**: $E \to \mathbb{R}$, and assume that G contains no negative-weight cycles reachable from the source vertex $s \in V$, so that shortest paths are well-defined.
- A ***shortest-paths tree*** rooted at s is a **directed subgraph $\boldsymbol{G' = (V', E')}$**, where $V' \subseteq V$ and $E' \subseteq E$, such that
  - V' is the set of vertices reachable from s in G,
  - G' forms a rooted tree with root s, and
  - for all $v \in V'$, the unique simple path from s to v in G' is a shortest path from s to v in G.



Shaded edges in (b) and (c) compose the sets of edges of a shortest path tree of the graph in (a).

THE TECHNIQUE OF COMPUTING A SHORTEST PATH

- The ***relaxation*** technique
- The process of ***relaxing*** an edge (u, v) consists of ***testing*** whether we can improve the shortest path to v found so far by going through u and, if so, ***updating*** $\boldsymbol{v.d}$ and $\boldsymbol{v.\pi}$.
  - For each vertex $v \in V$, we maintain an attribute $\boldsymbol{v.d}$, which is ***an upper bound*** on the weight of a shortest path from a given source vertex s to v.
  - We call $v.d$ a ***shortest-path estimate***.
  - **$\boldsymbol{v.\pi}$ is the predecessor of node $\boldsymbol{v}$.**
- The name ***relaxation*** might be ambiguous as we are not relaxing a path by stretching the path. Instead, we are ***relaxing a path*** by replacing it with a shorter path.

THE RELAXATION TECHNIQUE SUBROUTINES

- INITIALIZE-SINGLE-SOURCE (G, s)
  - Input
    - Graph $G = (V, E)$, represented by either adjacency list or adjacency matrix.
    - A source vertex s.
- RELAX (u, v, w)
  - Input
    - End vertex of v of path $p =< s, \dots, v >$ to be relaxed.
    - The vertex u that can potentially relax the path $p =< s, \dots, v >$.

INITIALIZE-SINGLE-SOURCE $(\boldsymbol{G}, \boldsymbol{s})$
for each vertex $v \in G.V$
$v.d = \infty$
$v.\pi =$ NIL
$s.d = 0$

RELAX $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$
if $v.d > u.d + w(u, v)$
$v.d = u.d + w(u, v)$
$v.\pi = u$

THE RELAXATION TECHNIQUE
THE VERTEX OBJECT

- For each vertex $\boldsymbol{v} \in V$,
  - $\boldsymbol{v.d}$ – ***an upper bound*** on the weight of a shortest path from a given source vertex s to v.
    - We call $\boldsymbol{v.d}$ a ***shortest-path estimate***.
  - $\boldsymbol{v.\pi}$ – the ***predecessor*** of vertex v.
    - If v has no predecessor, then $v.\pi = \mathtt{NIL}$.
      - For example, the source vertex $s.\pi =$ NIL.

INITIALIZE-SINGLE-SOURCE $(\boldsymbol{G}, \boldsymbol{s})$
for each vertex $v \in G.V$
$v.d = \infty$
$v.\pi =$ NIL
$s.d = 0$

RELAX $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$
if $v.d > u.d + w(u, v)$
$v.d = u.d + w(u, v)$
$v.\pi = u$

THE RELAXATION TECHNIQUE THE RELAX SUBROUTINE

- The **RELAX** $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$ subroutine examines vertex u that can potentially relax the path from s to v.
  - Let $p = < s, \dots, v >$ be the current shortest path from s to v, then $w(p)$ is mapped on to $v.d$ in the codes.
  - Let $p' =< s, \dots, u, v >$ be another path from s to v through vertex u, then $w(p')$ is mapped on to $u.d + w(u, v)$ in the codes.
  - Line 1 is comparing $w(p)$ with $w(p')$.
    - If going through the vertex u can shorten the path from s to v (condition being met), then relax the path in line 2 and 3.

INITIALIZE-SINGLE-SOURCE $(\boldsymbol{G}, \boldsymbol{s})$	
1	**for** each vertex $v \in G.V$
2	$\quad v.d = \infty$
3	$\quad v.\pi =$ NIL
4	$s.d = 0$

RELAX $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$	
1	**if** $v.d > u.d + w(u, v)$
2	$\quad v.d = u.d + w(u, v)$
3	$\quad v.\pi = u$

THE RELAXATION TECHNIQUE
RUNNING TIME

- INITIALIZE-SINGLE-SOURCE (G, s)
  - The running time $\Theta(|V|)$.
- RELAX (u, v, w)
  - The running time of the relaxation procedure is $\Theta(1)$.

INITIALIZE-SINGLE-SOURCE $(\boldsymbol{G}, \boldsymbol{s})$
for each vertex $v \in G.V$
$\quad v.d = \infty$
$\quad v.\pi =$ NIL
$s.d = 0$

RELAX $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$
if $v.d > u.d + w(u, v)$
$\quad v.d = u.d + w(u, v)$
$\quad v.\pi = u$

UP NEXT
SHORTEST-PATH ALGORITHMS

- Dijkstra’s algorithm
- Bellman-ford algorithm

- Both use the relaxation technique.

DIJKSTRA'S SHORTEST PATH ALGORITHM

- Dijkstra's algorithm solves the **single-source shortest paths** problem on a ***weighted***, ***directed*** graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ for the case in which all edge weights are non-negative.
  - Therefore, we assume that $w(u, v) \geq 0$ for each edge $(u, v) \in E$.

- We shall closely examine the algorithm in the following slides.

DIJKSTRA'S ALGORITHM
INPUT

- Graph $\boldsymbol{G} = (V, E)$ represented by adjacency ***lists***.
- The weight function $\boldsymbol{w}: E \rightarrow \mathbb{R}$
- The source vertex $\boldsymbol{s}$
- The *vertex* object
  - $\boldsymbol{v.d}$ – ***the shortest-path estimate***.
  - $\boldsymbol{v.\pi}$ – the ***predecessor*** of vertex v.

DIJKSTRA $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$
INITIALIZE-SINGLE-SOURCE (G, s)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
$u =$ EXTRACT-MIN (Q)
$S = S \cup \{u\}$
for each vertex $v \in \boldsymbol{G.Adj[u]}$
RELAX (u, v, w)

DIJKSTRA'S ALGORITHM
DATA STRUCTURE

- **Min-priority queue** $\boldsymbol{Q}$ **keyed by the** $\boldsymbol{d}$ **attribute** of a vertex.
  - EXTRACT-MIN (Q) **dequeues** the vertex that has the smallest value for d.

DIJKSTRA $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$
INITIALIZE-SINGLE-SOURCE (G, s)
$S = \emptyset$
$\boldsymbol{Q = G.V}$
while $Q \neq \emptyset$
$\boldsymbol{u} =$ **EXTRACT-MIN** $(\boldsymbol{Q})$
$S = S \cup \{u\}$
for each vertex $v \in G.Adj[u]$
RELAX (u, v, w)

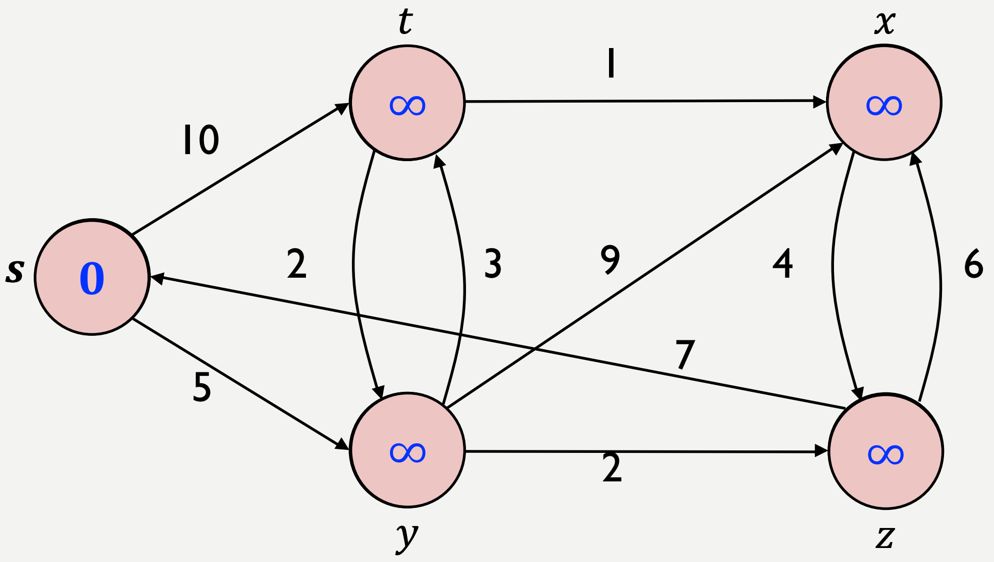
DIJKSTRA'S ALGORITHM
IDEA

- The algorithm repeatedly
  - **selects** the vertex $u \in V - S$ with the ***minimum*** shortest-path estimate
  - **adds** u to S
  - ***relaxes*** all edges leaving u

DIJKSTRA $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$
INITIALIZE-SINGLE-SOURCE (G, s)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
$\boldsymbol{u} =$ **EXTRACT-MIN** $(\boldsymbol{Q})$
$\boldsymbol{S} = \boldsymbol{S} \cup \{\boldsymbol{u}\}$
for each vertex $v \in G.Adj[u]$
RELAX (u, v, w)

DIJKSTRA'S ALGORITHM
IN ACTION - INITIALIZATION

- Apply the DIJKSTRA algorithm on the graph.



Adjacency lists

$s \rightarrow t \rightarrow y$

$t \rightarrow x \rightarrow y$

$x \rightarrow z$

$y \rightarrow t \rightarrow x \rightarrow z$

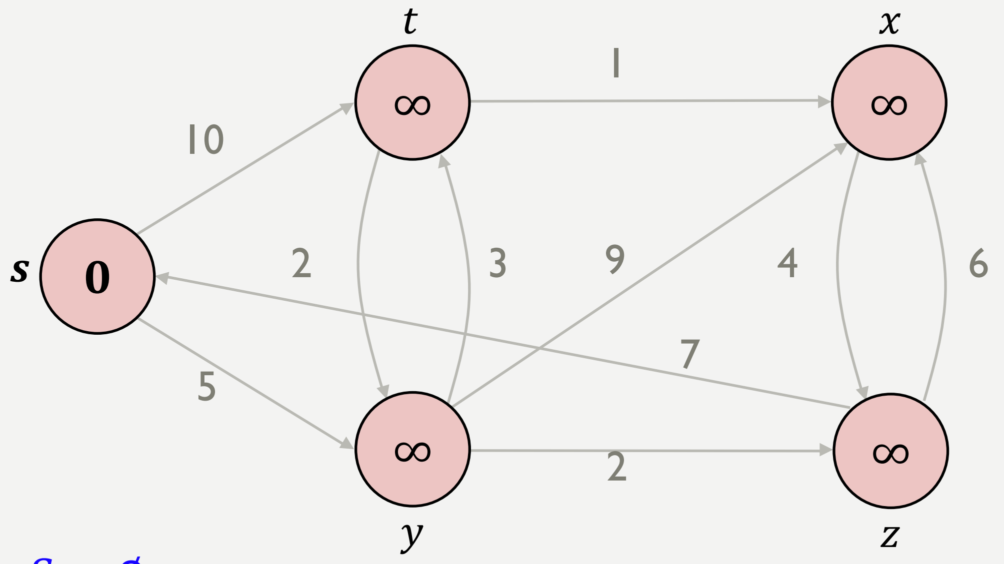
$z \rightarrow s \rightarrow x$

DIJKSTRA $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$
INITIALIZE-SINGLE-SOURCE (G, s)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
$u =$ EXTRACT-MIN (Q)
$S = S \cup \{u\}$
for each vertex $v \in G.Adj[u]$
RELAX (u, v, w)

INITIALIZE-SINGLE-SOURCE $(\boldsymbol{G}, \boldsymbol{s})$
for each vertex $v \in G.V$
$v.d = \infty$
$v.\pi =$ NIL
$s.d = 0$

DIJKSTRA'S ALGORITHM
IN ACTION – PRIOR TO WHILE

- Apply the DIJKSTRA algorithm on the graph.



Adjacency lists

$s \to t \to y$

$t \to x \to y$

$x \to z$

$y \to t \to x \to z$

$z \to s \to x$

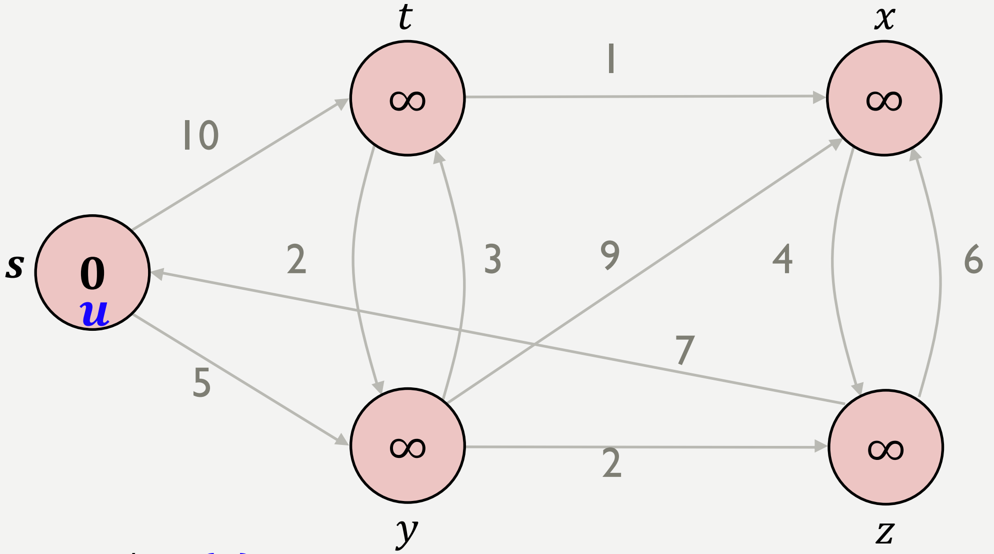
DIJKSTRA $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$
INITIALIZE-SINGLE-SOURCE (G, s)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
$u =$ EXTRACT-MIN (Q)
$S = S \cup \{u\}$
for each vertex $v \in G.Adj[u]$
RELAX (u, v, w)

$S = \emptyset$

$Q = \{s, t, x, y, z\}$ // all vertexes but s have infinte shortest-path estimate, order them alphabetically

DIJKSTRA'S ALGORITHM IN ACTION – WHILE

- Apply the DIJKSTRA algorithm on the graph.



Adjacency lists

$s \to t \to y$

$t \to x \to y$

$x \to z$

$y \to t \to x \to z$

$z \to s \to x$

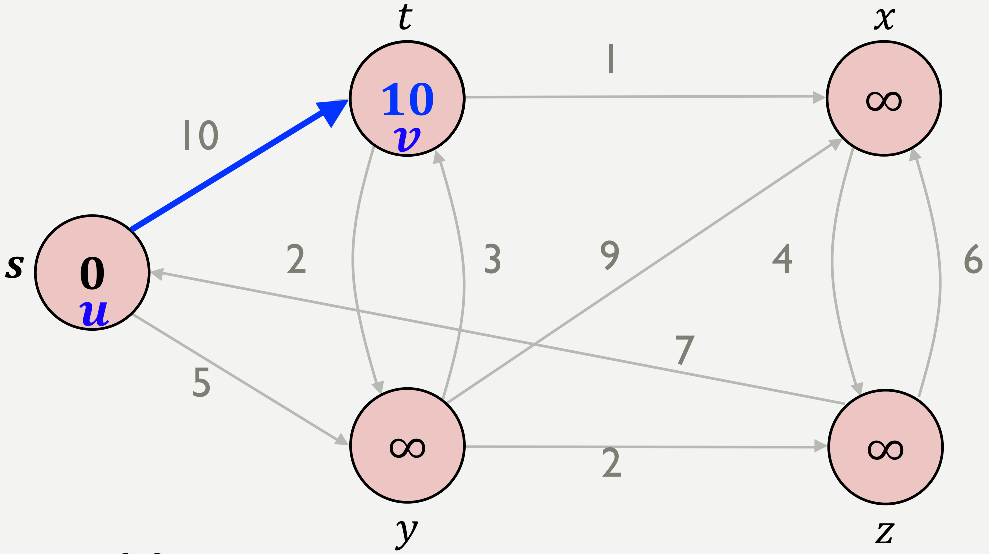
DIJKSTRA $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$
INITIALIZE-SINGLE-SOURCE (G, s)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
$u =$ EXTRACT-MIN (Q)
$S = S \cup \{u\}$
for each vertex $v \in G.Adj[u]$
RELAX (u, v, w)

$S = \emptyset \cup \{s\}$

$Q = \{\cancel{s}, t, x, y, z\}$ // all vertexes but s have infinte shortest-path estimate, order them alphabetically

DIJKSTRA'S ALGORITHM
IN ACTION – WHILE

- Apply the DIJKSTRA algorithm on the graph.



$S = \{s\}$

$Q = \{t, x, y, z\}$ $\implies Q = \{t, x, y, z\}$!!!

Adjacency lists

$s \to \boxed{t} \to y$

$t \to x \to y$

$x \to z$

$y \to t \to x \to z$

$z \to s \to x$

DIJKSTRA $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$

INITIALIZE-SINGLE-SOURCE (G, s)

$S = \emptyset$

$Q = G.V$

while $Q \neq \emptyset$

  $u =$ EXTRACT-MIN (Q)

  $S = S \cup \{u\}$

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

    RELAX (u, v, w)

RELAX $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$

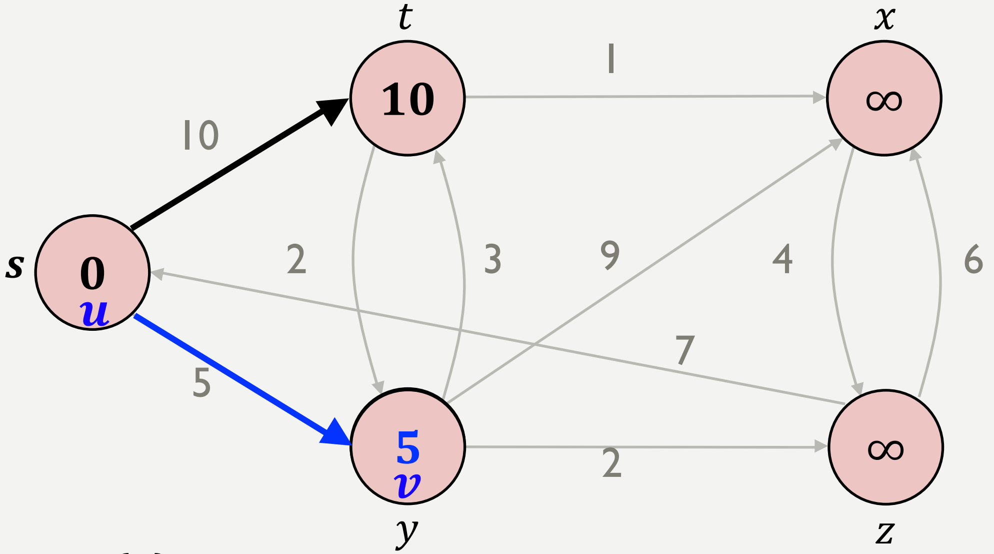
if $v.d > u.d + w(u, v)$ $\infty > \mathbf{0} + \mathbf{10}$

  $v.d = u.d + w(u, v)$

  $v.\pi = u$

DIJKSTRA'S ALGORITHM
IN ACTION – WHILE

- Apply the DIJKSTRA algorithm on the graph.



$S = \{s\}$

$Q = \{t, x, y, z\}$ $\Longrightarrow Q = \{y, t, x, z\}$!!!

Adjacency lists

$s \to t \to \boxed{y}$

$t \to x \to y$

$x \to z$

$y \to t \to x \to z$

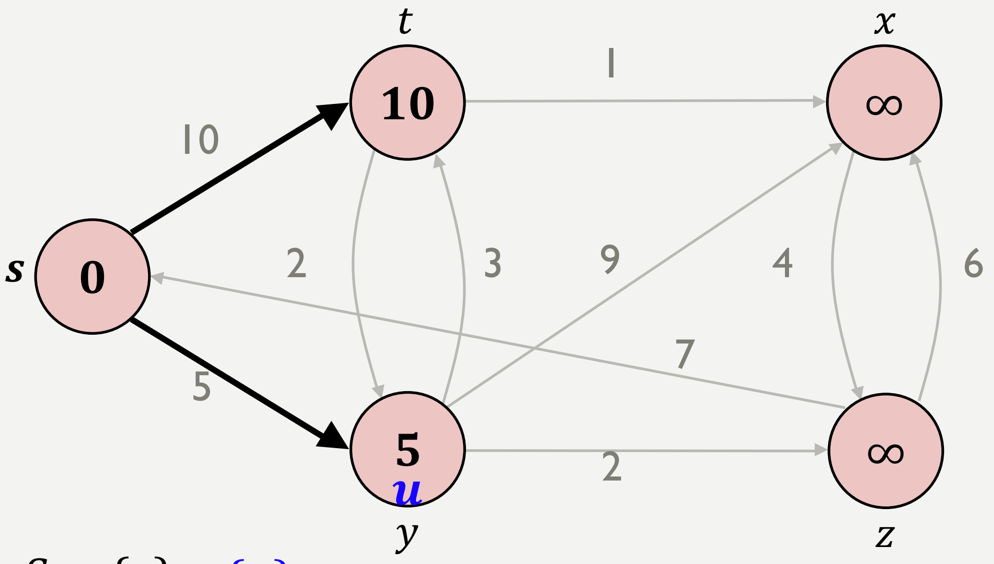
$z \to s \to x$

DIJKSTRA $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$
INITIALIZE-SINGLE-SOURCE (G, s)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
$u =$ EXTRACT-MIN (Q)
$S = S \cup \{u\}$
for each vertex $v \in G.Adj[u]$
RELAX (u, v, w)

RELAX $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$
if $v.d > u.d + w(u, v)$ $\infty > \mathbf{0} + \mathbf{5}$
$v.d = u.d + w(u, v)$
$v.\pi = u$

DIJKSTRA'S ALGORITHM
IN ACTION – WHILE

- Apply the DIJKSTRA algorithm on the graph.



$S = \{s\} \cup \{y\}$

$Q = \{\cancel{y}, t, x, z\}$

Adjacency lists

$s \to t \to y$

$t \to x \to y$

$x \to z$

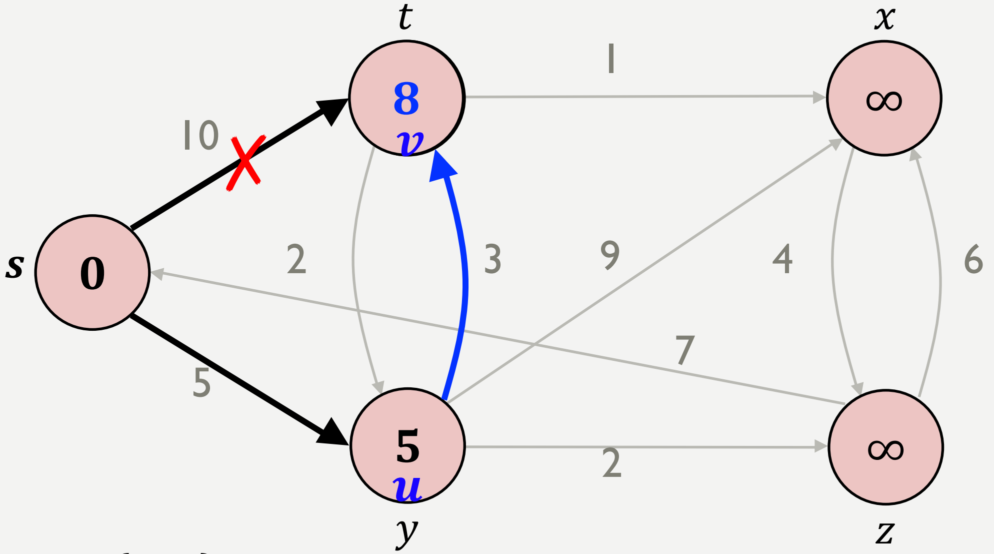
$\boxed{y} \to t \to x \to z$

$z \to s \to x$

DIJKSTRA $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$
INITIALIZE-SINGLE-SOURCE (G, s)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
$u =$ EXTRACT-MIN (Q)
$S = S \cup \{u\}$
for each vertex $v \in G.Adj[u]$
RELAX (u, v, w)

DIJKSTRA'S ALGORITHM
IN ACTION – WHILE

- Apply the DIJKSTRA algorithm on the graph.



Adjacency lists

$s \to t \to y$

$t \to x \to y$

$x \to z$

$y \to \boxed{t} \to x \to z$

$z \to s \to x$

DIJKSTRA $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$

INITIALIZE-SINGLE-SOURCE (G, s)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
    $u =$ EXTRACT-MIN (Q)
    $S = S \cup \{u\}$
    for each vertex $v \in G.Adj[u]$
        RELAX (u, v, w)

RELAX $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$

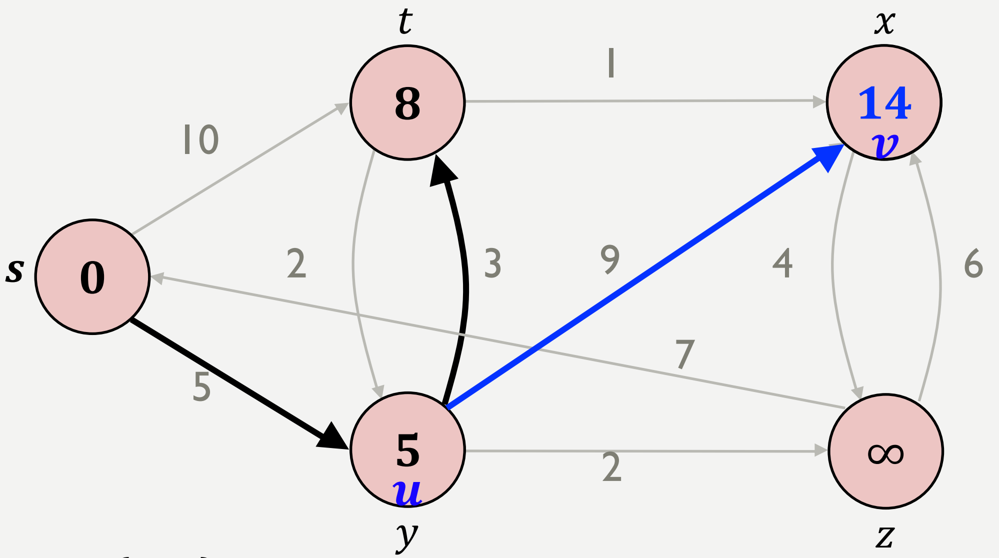
if $v.d > u.d + w(u, v)$ $\mathbf{10 > 5 + 3}$
    $v.d = u.d + w(u, v)$
    $v.\pi = u$

$S = \{s, y\}$ // in the order of them being added to the set

$Q = \{t, x, z\}$ $\Longrightarrow Q = \{t, x, z\}$!!!

DIJKSTRA'S ALGORITHM
IN ACTION – WHILE

- Apply the DIJKSTRA algorithm on the graph.



$S = \{s, y\}$

$Q = \{t, x, z\}$ $\quad \Longrightarrow Q = \{t, x, z\}$!!!

Adjacency lists

$s \to t \to y$

$t \to x \to y$

$x \to z$

$y \to t \to \boxed{x} \to z$

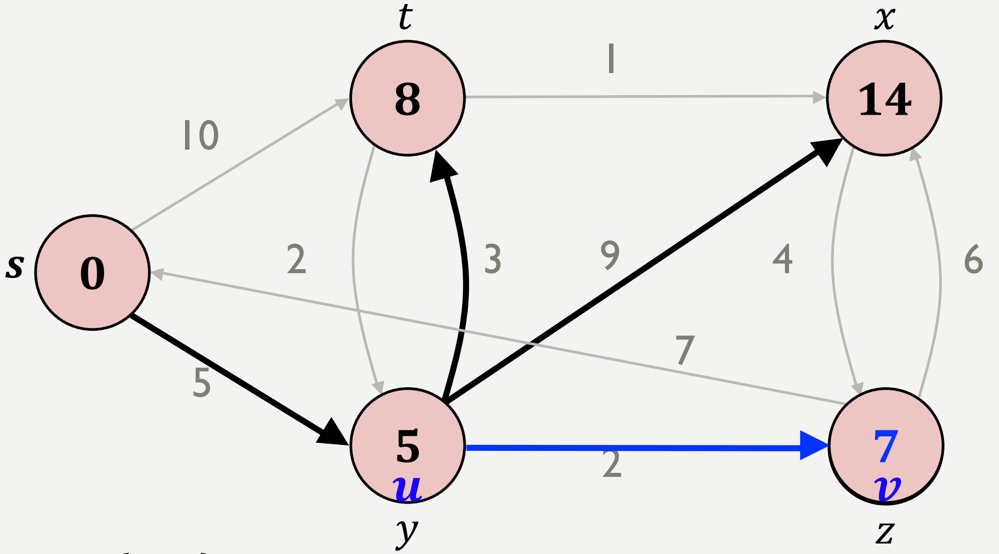
$z \to s \to x$

DIJKSTRA $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$
INITIALIZE-SINGLE-SOURCE (G, s)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
$\quad u =$ EXTRACT-MIN (Q)
$\quad S = S \cup \{u\}$
$\quad$ **for** each vertex $v \in G.Adj[u]$
$\qquad$ RELAX (u, v, w)

RELAX $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$
if $v.d > u.d + w(u, v)$ $\quad \boldsymbol{\infty > 5 + 9}$
$\quad v.d = u.d + w(u, v)$
$\quad v.\pi = u$

DIJKSTRA'S ALGORITHM
IN ACTION – WHILE

- Apply the DIJKSTRA algorithm on the graph.



$S = \{s, y\}$

$Q = \{t, x, z\}$ $\Longrightarrow Q = \{z, t, x\}$!!!

Adjacency lists

$s \to t \to y$

$t \to x \to y$

$x \to z$

$y \to t \to x \boxed{\to z}$

$z \to s \to x$

DIJKSTRA $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$

INITIALIZE-SINGLE-SOURCE (G, s)

$S = \emptyset$

$Q = G.V$

while $Q \neq \emptyset$

  $u =$ EXTRACT-MIN (Q)

  $S = S \cup \{u\}$

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

    RELAX (u, v, w)

RELAX $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$

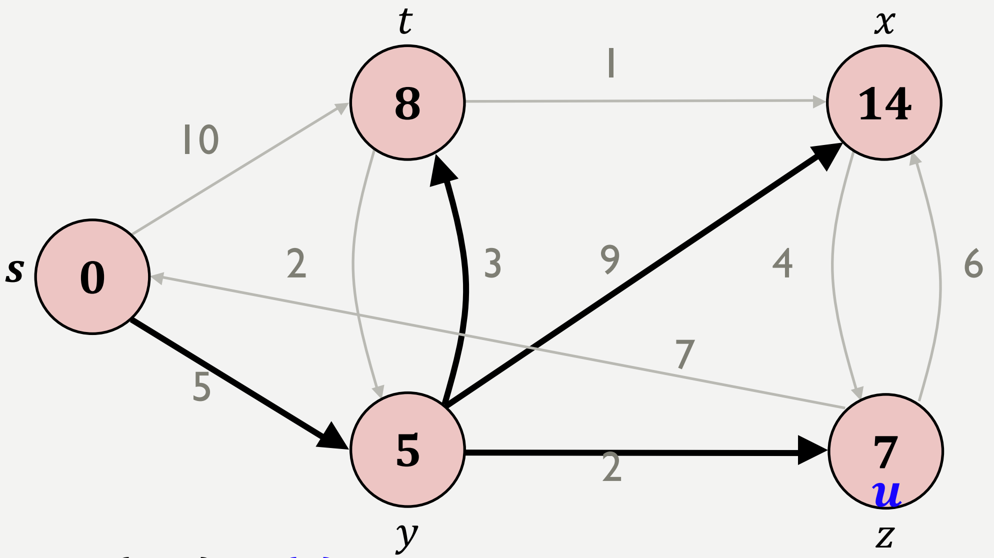
if $v.d > u.d + w(u, v)$ $\infty > \mathbf{5} + \mathbf{2}$

  $v.d = u.d + w(u, v)$

  $v.\pi = u$

DIJKSTRA'S ALGORITHM
IN ACTION – WHILE

- Apply the DIJKSTRA algorithm on the graph.



$S = \{s, y\} \cup \{z\}$

$Q = \{\cancel{z}, t, x\}$

Adjacency lists

$s \rightarrow t \rightarrow y$

$t \rightarrow x \rightarrow y$

$x \rightarrow z$

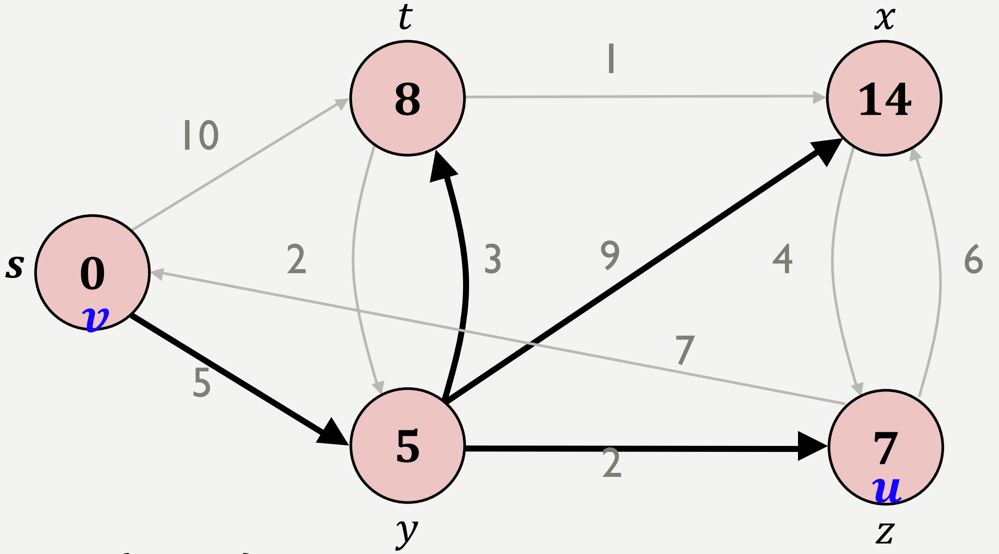
$y \rightarrow t \rightarrow x \rightarrow z$

$z \rightarrow s \rightarrow x$

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

DIJKSTRA'S ALGORITHM
IN ACTION – WHILE

- Apply the DIJKSTRA algorithm on the graph.



$S = \{s, y, z\}$ // in the order of them being added to the set

$Q = \{t, x\}$

Adjacency lists

$s \to t \to y$

$t \to x \to y$

$x \to z$

$y \to t \to x \to z$

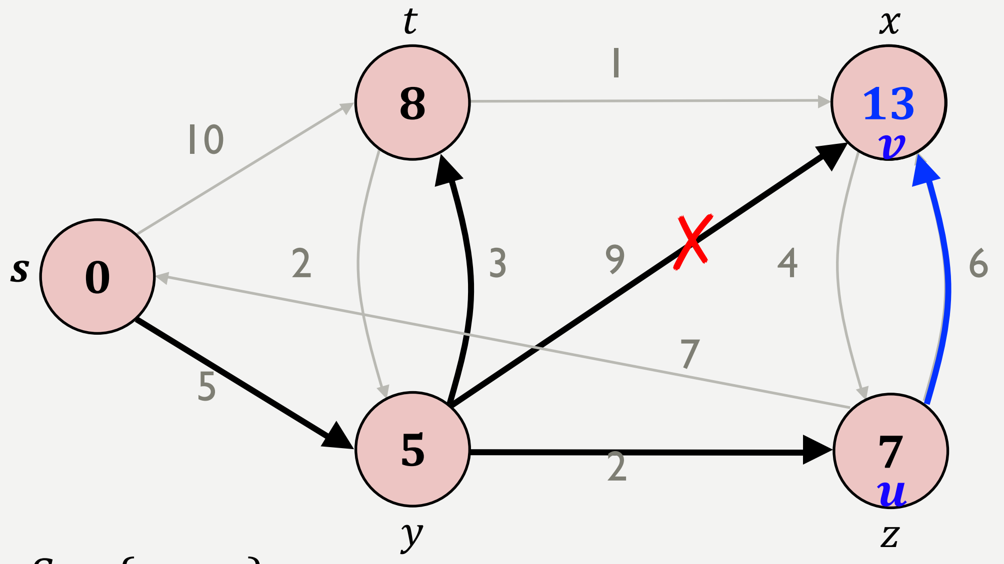
$z \to \boxed{s} \to x$

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

RELAX $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$	
1	**if** $v.d > u.d + w(u, v)$ $\mathbf{0} \not> \mathbf{7} + \mathbf{7}$
2	$v.d = u.d + w(u, v)$
3	$v.\pi = u$

DIJKSTRA'S ALGORITHM IN ACTION – WHILE

- Apply the DIJKSTRA algorithm on the graph.



$S = \{s, y, z\}$

$Q = \{t, x\}$ $\Longrightarrow Q = \{t, x\}$!!!

Adjacency lists

$s \rightarrow t \rightarrow y$

$t \rightarrow x \rightarrow y$

$x \rightarrow z$

$y \rightarrow t \rightarrow x \rightarrow z$

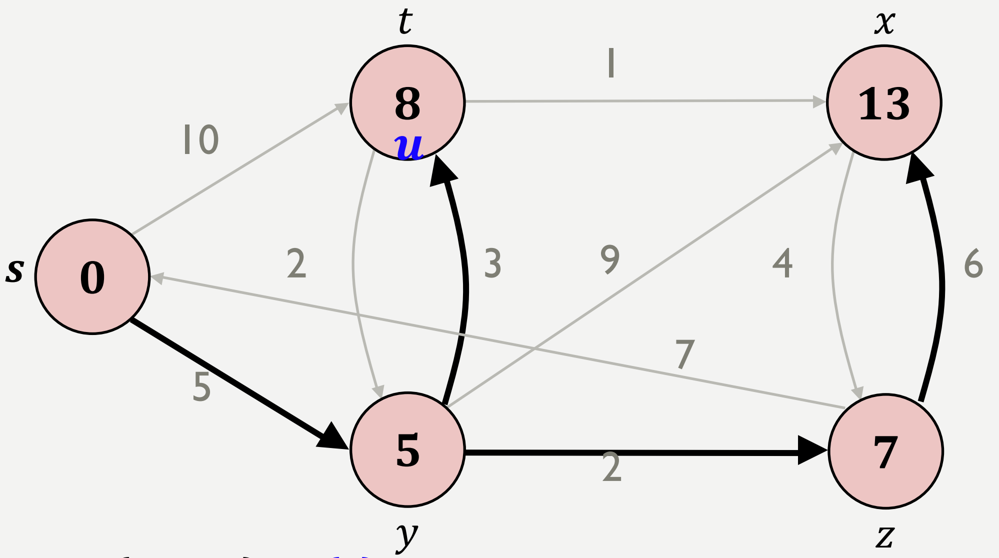
$z \rightarrow s \boxed{\rightarrow x}$

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

RELAX (u, v, w)	
1	**if** $v.d > u.d + w(u, v)$ $\mathbf{14 > 7 + 6}$
2	$v.d = u.d + w(u, v)$
3	$v.\pi = u$

DIJKSTRA'S ALGORITHM
IN ACTION – WHILE

- Apply the DIJKSTRA algorithm on the graph.



$S = \{s, y, z\} \cup \{t\}$

$Q = \{\cancel{t}, x\}$

Adjacency lists

$s \to t \to y$

$t \to x \to y$

$x \to z$

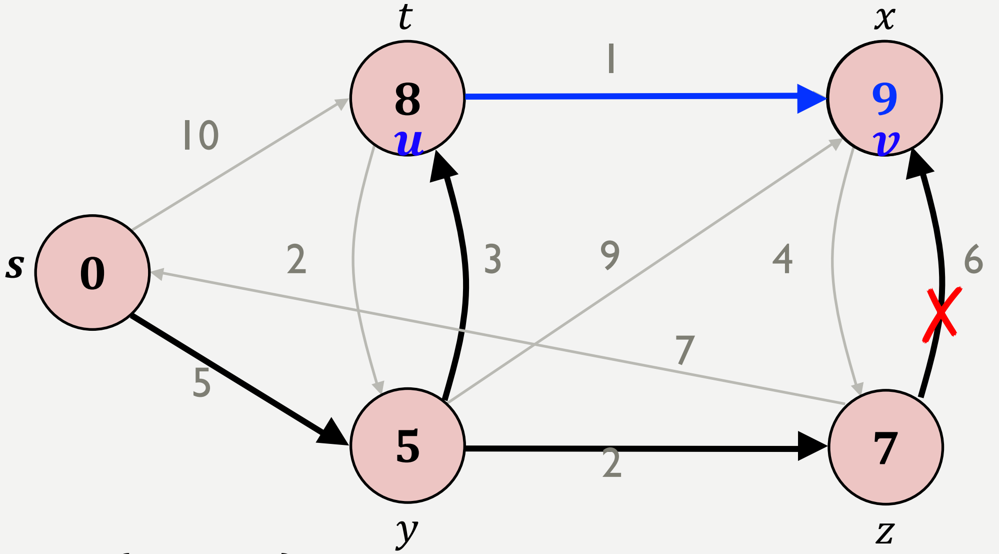
$y \to t \to x \to z$

$z \to s \to x$

DIJKSTRA $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$
INITIALIZE-SINGLE-SOURCE (G, s)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
$u =$ EXTRACT-MIN (Q)
$S = S \cup \{u\}$
for each vertex $v \in G.Adj[u]$
RELAX (u, v, w)

DIJKSTRA'S ALGORITHM
IN ACTION – WHILE

- Apply the DIJKSTRA algorithm on the graph.



$S = \{s, y, z, t\}$

$Q = \{x\} \Longrightarrow Q = \{x\}$!!!

Adjacency lists

$s \rightarrow t \rightarrow y$

$t \rightarrow \boxed{x} \rightarrow y$

$x \rightarrow z$

$y \rightarrow t \rightarrow x \rightarrow z$

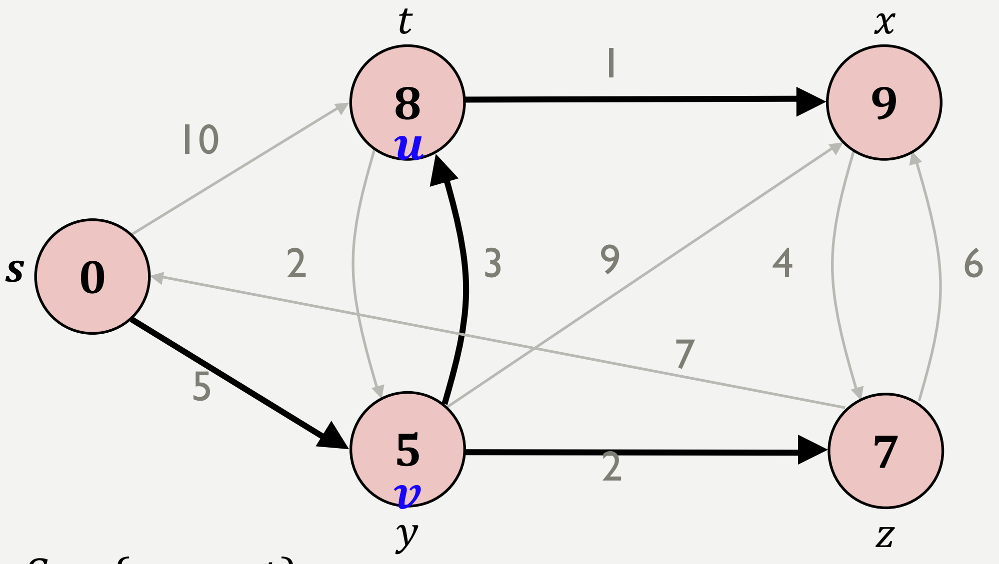
$z \rightarrow s \rightarrow x$

DIJKSTRA (G, w, s)
INITIALIZE-SINGLE-SOURCE (G, s)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
$\quad u =$ EXTRACT-MIN (Q)
$\quad S = S \cup \{u\}$
$\quad$ **for** each vertex $v \in G.Adj[u]$
$\quad\quad$ RELAX (u, v, w)

RELAX (u, v, w)
if $v.d > u.d + w(u, v)$ $\mathbf{13 > 8 + 1}$
$\quad v.d = u.d + w(u, v)$
$\quad v.\pi = u$

DIJKSTRA'S ALGORITHM
IN ACTION – WHILE

- Apply the DIJKSTRA algorithm on the graph.



$S = \{s, y, z, t\}$

$Q = \{x\}$

Adjacency lists

$s \to t \to y$

$t \to x \to \boxed{y}$

$x \to z$

$y \to t \to x \to z$

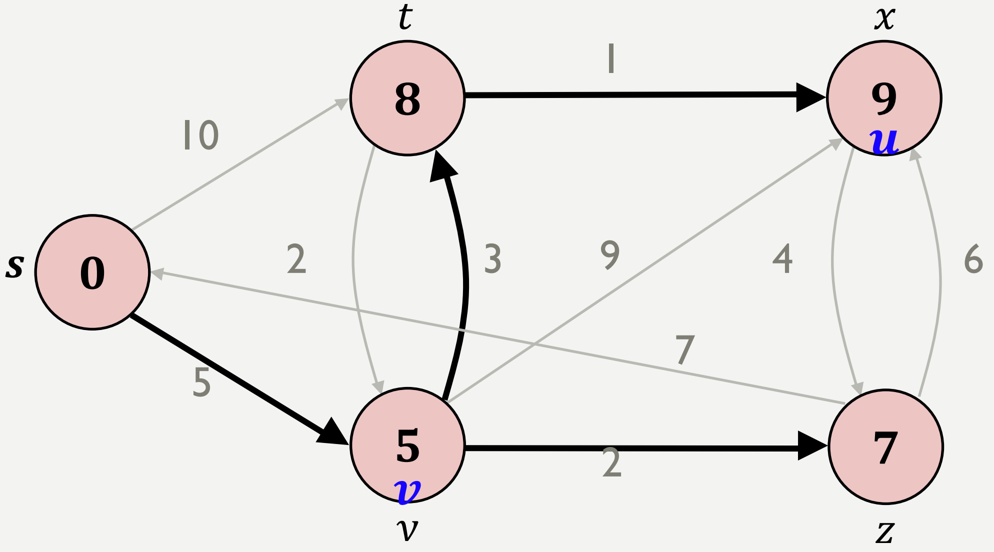
$z \to s \to x$

DIJKSTRA $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$
INITIALIZE-SINGLE-SOURCE (G, s)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
$u =$ EXTRACT-MIN (Q)
$S = S \cup \{u\}$
for each vertex $v \in G.Adj[u]$
RELAX (u, v, w)

RELAX $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$
if $v.d > u.d + w(u, v)$ $\quad \mathbf{5} \not> \mathbf{8} + \mathbf{2}$
$v.d = u.d + w(u, v)$
$v.\pi = u$

DIJKSTRA'S ALGORITHM
IN ACTION – WHILE

- Apply the DIJKSTRA algorithm on the graph.



$S = \{s, y, z, t\} \cup \{x\}$

$Q = \{\cancel{x}\}$

Adjacency lists

$s \to t \to y$

$t \to x \to y$

$x \to z$

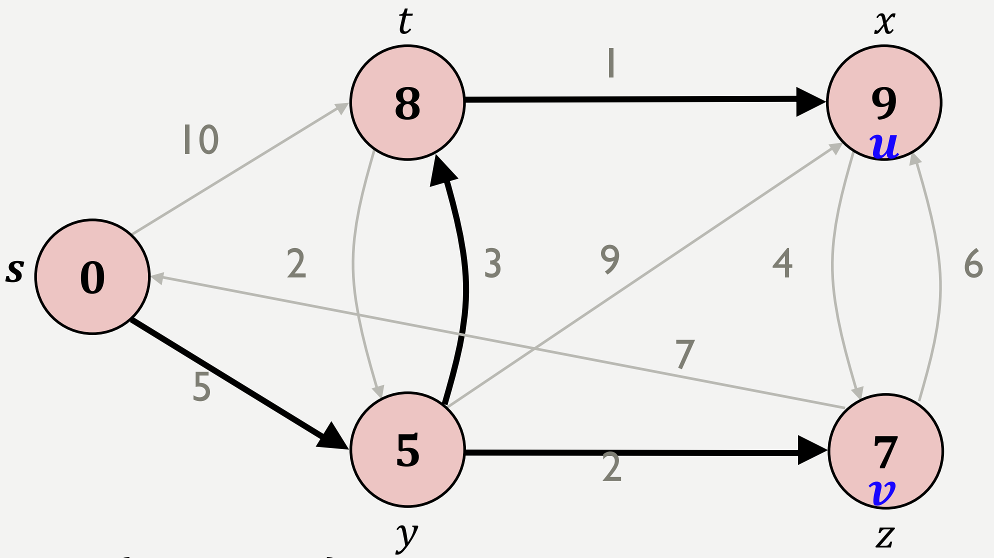
$y \to t \to x \to z$

$z \to s \to x$

DIJKSTRA $(\boldsymbol{G, w, s})$
INITIALIZE-SINGLE-SOURCE (G, s)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
$u =$ EXTRACT-MIN (Q)
$S = S \cup \{u\}$
for each vertex $v \in G.Adj[u]$
RELAX (u, v, w)

DIJKSTRA'S ALGORITHM
IN ACTION – WHILE

- Apply the DIJKSTRA algorithm on the graph.



$S = \{s, y, z, t, x\}$ // in the order of them being added to the set

$Q = \emptyset$ Stop!

Adjacency lists

$s \to t \to y$

$t \to x \to y$

$x \to \boxed{z}$

$y \to t \to x \to z$

$z \to s \to x$

DIJKSTRA $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$

INITIALIZE-SINGLE-SOURCE (G, s)

$S = \emptyset$

$Q = G.V$

while $Q \neq \emptyset$

$u =$ EXTRACT-MIN (Q)

$S = S \cup \{u\}$

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

RELAX (u, v, w)

RELAX $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$

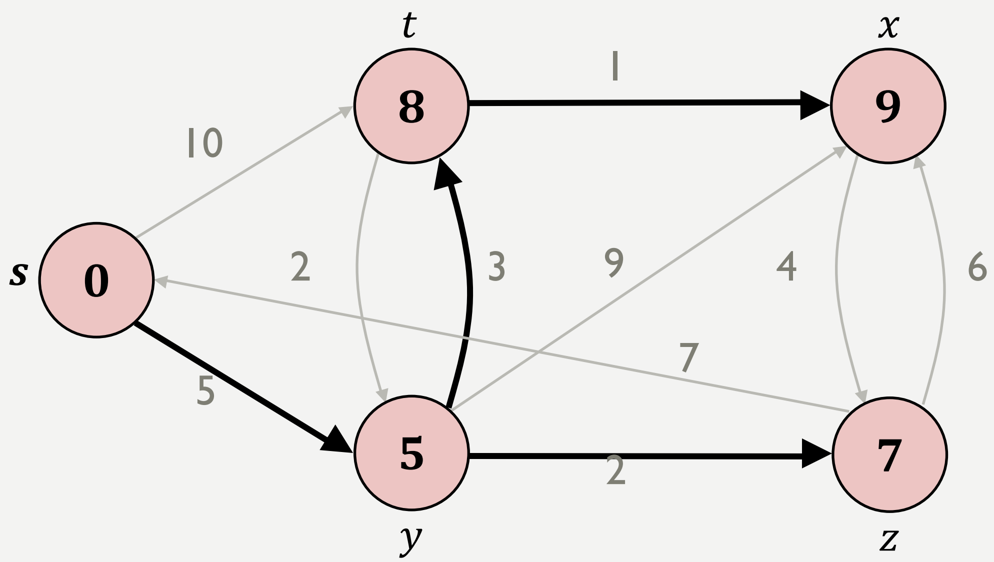
if $v.d > u.d + w(u, v)$ $\mathbf{7 \ngtr 9 + 4}$

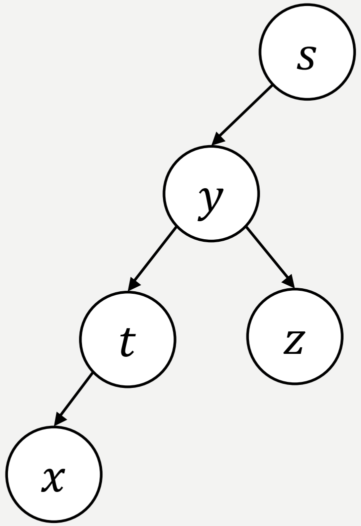
$v.d = u.d + w(u, v)$

$v.\pi = u$

DIJKSTRA'S ALGORITHM
SHORTEST-PATH TREE

- The DIJKSTRA algorithm generates a ***shortest-path tree of the original graph G.***





Tree visualization

DIJKSTRA'S ALGORITHM RUNNING TIME

- The **INITIALIZATION** costs $\Theta($ _____ $)$.
- Min-priority queue operation
  - Let f be the cost function of **building** Q
  - Let g be the cost function of **EXTRACT-MIN** algorithm
  - Let s be the cost function of **DECREASE-KEY** operation.
  - Each vertex is extracted from Q ____ time(s)
- The running time of DIJKSTRA algorithm is

$$T = \underbrace{\_\_\_\_\_\_\_\_}_{\text{Initialization.}} + \underbrace{\_\_\_\_\_\_\_\_}_{\text{Cost of building } Q} + \Theta(|V|) \cdot \underbrace{\_\_\_\_\_\_\_\_}_{\text{Cost of EXTRAC-MIN}} + \Theta(|E|) \cdot \underbrace{\_\_\_\_\_\_\_\_}_{\text{Cost of DECREASE-KEY}}$$

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

DIJKSTRA'S ALGORITHM
RUNNING TIME - INIT.

- The **INITIALIZATION** costs $\Theta($ _____ $)$.

DIJKSTRA $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$
INITIALIZE-SINGLE-SOURCE (G, s)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
$u =$ EXTRACT-MIN (Q)
$S = S \cup \{u\}$
for each vertex $v \in G.Adj[u]$
RELAX (u, v, w)

INITIALIZE-SINGLE-SOURCE $(\boldsymbol{G}, \boldsymbol{s})$
for each vertex $v \in G.V$
$v.d = \infty$
$v.\pi =$ NIL
$s.d = 0$

DIJKSTRA'S ALGORITHM
RUNNING TIME OF MAINTAINING Q

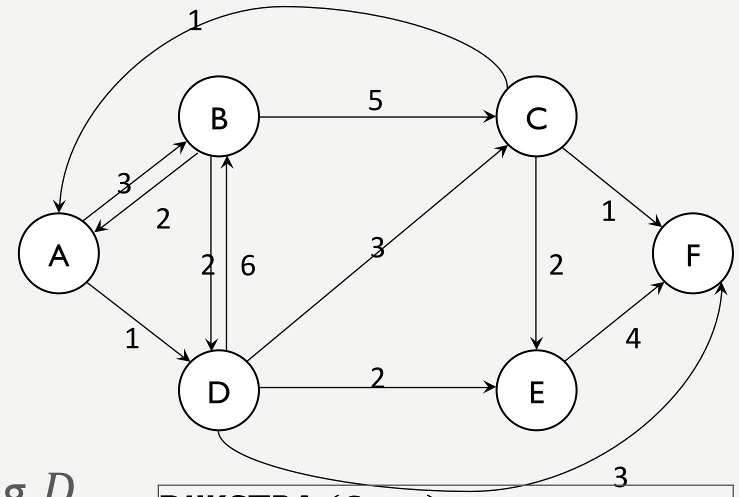
- Min-priority queue operation
  - Let f be the cost function of **building** Q
  - Let g be the cost function of **EXTRACT-MIN** algorithm
  - Let s be the cost function of **DECREASE-KEY** operation.
  - Each vertex is extracted from Q ____ time(s)

DIJKSTRA $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$
INITIALIZE-SINGLE-SOURCE (G, s)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
$\quad u =$ EXTRACT-MIN (Q)
$\quad S = S \cup \{u\}$
$\quad$ **for** each vertex $v \in G.Adj[u]$
$\quad\quad$ RELAX (u, v, w)

- The running time of DIJKSTRA algorithm is

$$T = \underbrace{\_\_\_\_\_\_\_\_}_{\text{Initialization.}} + \underbrace{\_\_\_\_\_\_\_\_}_{\text{Cost of building } Q} + \Theta(|V|) \cdot \underbrace{\_\_\_\_\_\_\_\_}_{\text{Cost of EXTRAC-MIN}} + \Theta(|E|) \cdot \underbrace{\_\_\_\_\_\_\_\_}_{\text{Cost of DECREASE-KEY}}$$

DIJKSTRA'S ALGORITHM
PRACTICE



- Run Dijkstra's algorithm on the graph with source vertex being D. Fill the table as you are "walking through" the algorithm
  - Fill ONLY the d attribute ***when the value is changed*** during that iteration/initialization.
  - Show the elements of S and Q at the beginning of an iteration.

DIJKSTRA $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$

INITIALIZE-SINGLE-SOURCE (G, s)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
    $u =$ EXTRACT-MIN (Q)
    $S = S \cup \{u\}$
    for each vertex $v \in G.Adj[u]$
        RELAX (u, v, w)

Iteration	Set S	Q	$A.d$	$B.d$	$C.d$	$D.d$	$E.d$	$F.d$
Init.	$\emptyset$	$\{D, A, B, C, E, F\}$	∞	∞	∞	0	∞	∞

UP NEXT
BELLMAN-FORD ALGORITHM

- The ***Bellman-Ford algorithm*** solves the **single-source shortest paths** problem on a ***weighted***, ***directed*** graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ with source s and weight function $\boldsymbol{w}: E \to \mathbb{R}$, where $w(u, v)$ **may be negative** for each edge $(u, v) \in E$.

- Returns a Boolean value indicating whether or not there is a ***negative-weight cycle*** that is reachable from the source.
  - If there is such a cycle, no solution

BELLMAN-FORD ALGORITHM

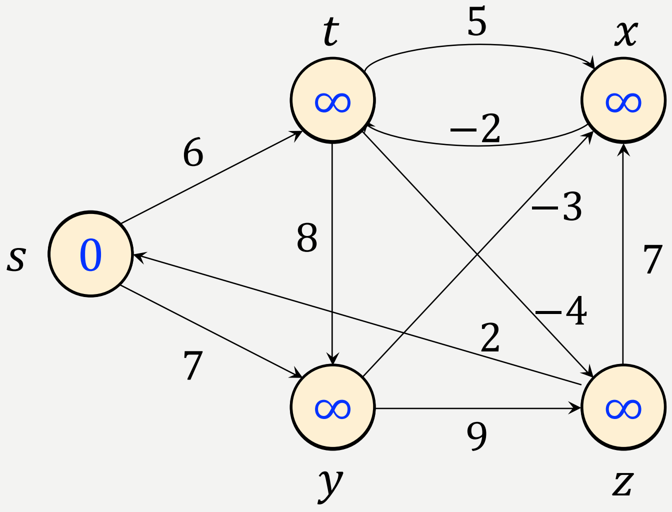
- Graph $\boldsymbol{G} = (V, E)$ represented by adjacency **lists**.
- The weight function $\boldsymbol{w}: E \to \mathbb{R}$
- The source vertex $\boldsymbol{s}$
- The vertex object
  - $\boldsymbol{v.d}$ – ***the shortest-path estimate***.
  - $\boldsymbol{v.\pi}$ – the ***predecessor*** of vertex v.

- No special data structure is used.

BELLMAN-FORD $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$
INITIALIZE-SINGLE-SOURCE (G, s)
for $i = 1$ **to** $\lvert G.V \rvert - 1$
for each edge $(u, v) \in G.E$
RELAX (u, v, w)
for each edge $(u, v) \in G.E$
if $v.d > u.d + w(u, v)$
return $FALSE$
return $TRUE$

BELLMAN-FORD ALGORITHM
IN ACTION

- Apply the BELLMAN-FORD algorithm on the graph.

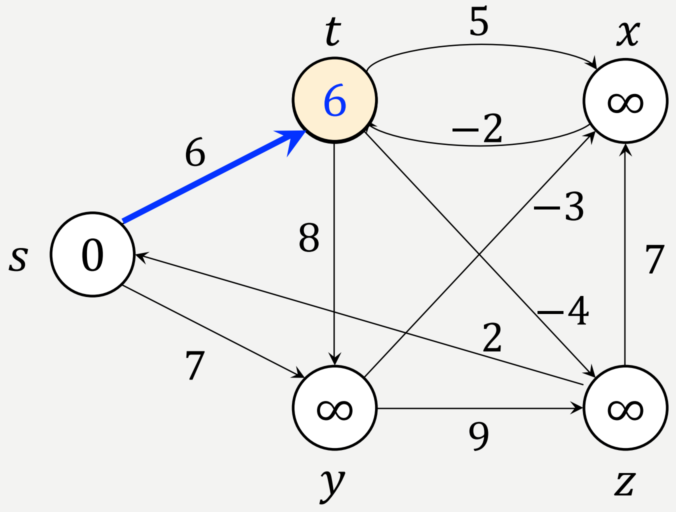


Adjacency list
$s \rightarrow t \rightarrow y$
$t \rightarrow x \rightarrow y \rightarrow z$
$x \rightarrow t$
$y \rightarrow x \rightarrow z$
$z \rightarrow s \rightarrow x$

BELLMAN-FORD $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$
INITIALIZE-SINGLE-SOURCE (G, s)
for $i = 1$ **to** $\lvert G.V \rvert - 1$
for each edge $(u, v) \in G.E$
RELAX (u, v, w)
for each edge $(u, v) \in G.E$
if $v.d > u.d + w(u, v)$
return $FALSE$
return $TRUE$

BELLMAN-FORD ALGORITHM IN ACTION

- Apply the BELLMAN-FORD algorithm on the graph.



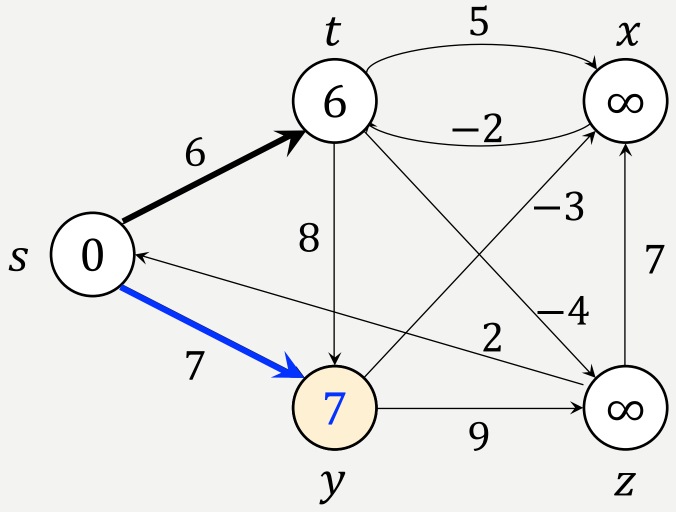
Adjacency list
$s \rightarrow \boldsymbol{t} \rightarrow y$
$t \rightarrow x \rightarrow y \rightarrow z$
$x \rightarrow t$
$y \rightarrow x \rightarrow z$
$z \rightarrow s \rightarrow x$

BELLMAN-FORD $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$
INITIALIZE-SINGLE-SOURCE (G, s)
for $i = 1$ **to** $|G.V| - 1$
  for each edge $(u, v) \in G.E$
    RELAX (u, v, w)
for each edge $(u, v) \in G.E$
  if $v.d > u.d + w(u, v)$
    return $FALSE$
return $TRUE$

Iteration	RELAX(u, v)	$t.d$	$x.d$	$y.d$	$z.d$
$i = 1$	(s, t)	6			

BELLMAN-FORD ALGORITHM
IN ACTION

- Apply the BELLMAN-FORD algorithm on the graph.



Adjacency list

$s \to t \to \boldsymbol{y}$

$t \to x \to y \to z$

$x \to t$

$y \to x \to z$

$z \to s \to x$

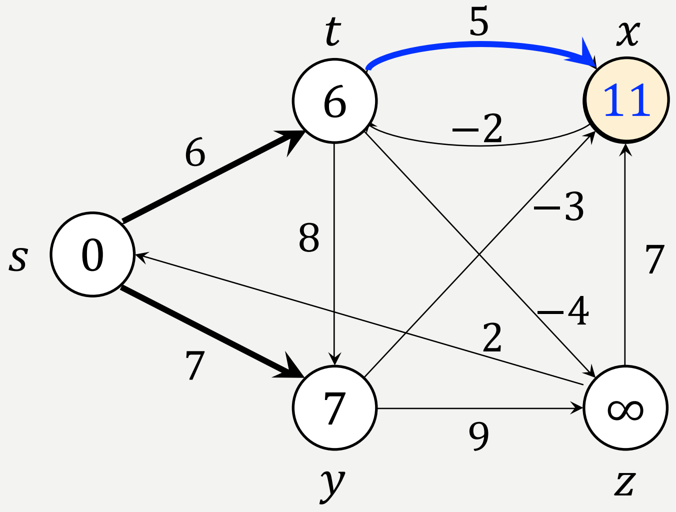
BELLMAN-FORD $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$

```
INITIALIZE-SINGLE-SOURCE (G, s)
for i = 1 to |G.V| − 1
        for each edge (u, v) ∈ G.E
                RELAX (u, v, w)
for each edge (u, v) ∈ G.E
        if v.d > u.d + w(u, v)
                return FALSE
return TRUE
```

Iteration	RELAX(u, v)	$t.d$	$x.d$	$y.d$	$z.d$
$i = 1$	(s, t)	6			
	(s, y)			7	

BELLMAN-FORD ALGORITHM
IN ACTION

- Apply the BELLMAN-FORD algorithm on the graph.



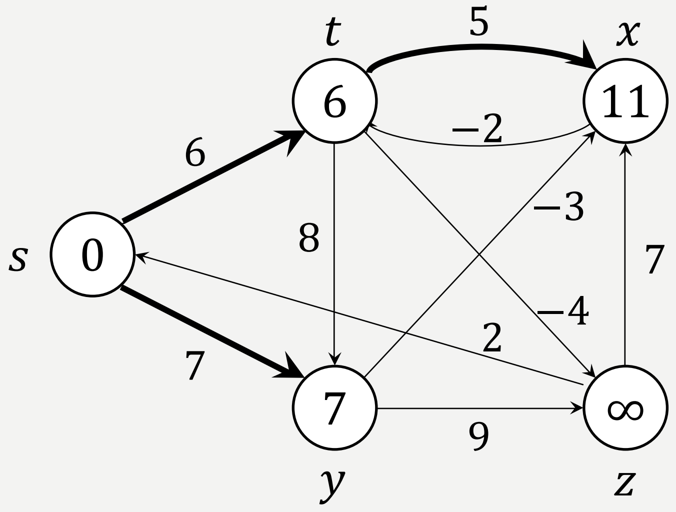
Adjacency list
$s \to t \to y$
$t \to \boldsymbol{x} \to y \to z$
$x \to t$
$y \to x \to z$
$z \to s \to x$

BELLMAN-FORD $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$
INITIALIZE-SINGLE-SOURCE (G, s)
for $i = 1$ **to** $|G.V| - 1$
    for each edge $(u, v) \in G.E$
        RELAX (u, v, w)
for each edge $(u, v) \in G.E$
    if $v.d > u.d + w(u, v)$
        return $FALSE$
return $TRUE$

Iteration	RELAX(u, v)	$t.d$	$x.d$	$y.d$	$z.d$
$i = 1$	(s, t)	6			
	(s, y)			7	
	(t, x)		11		

BELLMAN-FORD ALGORITHM
IN ACTION

- Apply the BELLMAN-FORD algorithm on the graph.



Adjacency list
$s \to t \to y$
$t \to x \to \boldsymbol{y} \to z$
$x \to t$
$y \to x \to z$
$z \to s \to x$

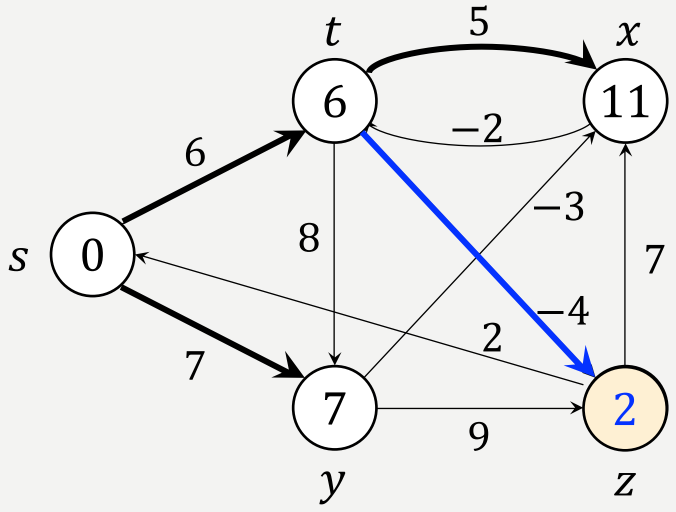
BELLMAN-FORD $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$

INITIALIZE-SINGLE-SOURCE (G, s)
for $i = 1$ **to** $\lvert G.V \rvert - 1$
for each edge $(u, v) \in G.E$
RELAX (u, v, w)
for each edge $(u, v) \in G.E$
if $v.d > u.d + w(u, v)$
return $FALSE$
return $TRUE$

Iteration	RELAX(u, v)	$t.d$	$x.d$	$y.d$	$z.d$
$i = 1$	(s, t)	6			
	(s, y)			7	
	(t, x)		11		
	(t, y)				

BELLMAN-FORD ALGORITHM
IN ACTION

- Apply the BELLMAN-FORD algorithm on the graph.



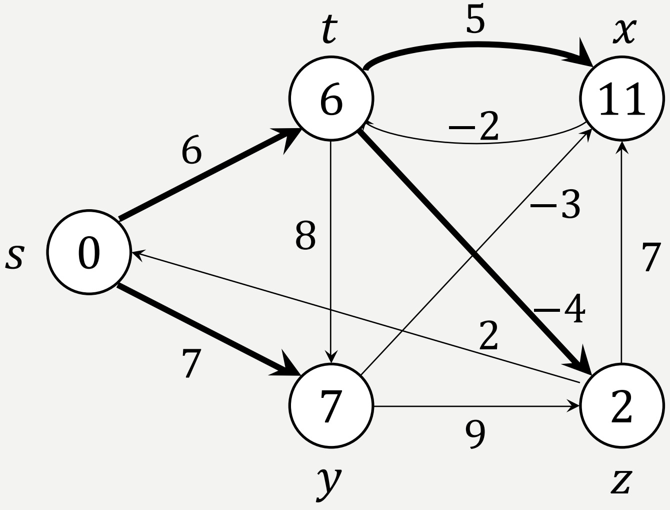
Adjacency list
$s \to t \to y$
$t \to x \to y \to \boldsymbol{z}$
$x \to t$
$y \to x \to z$
$z \to s \to x$

BELLMAN-FORD $(\boldsymbol{G, w, s})$	
1	INITIALIZE-SINGLE-SOURCE (G, s)
2	**for** $i = 1$ **to** $\lvert G.V \rvert - 1$
3	**for** each edge $(u, v) \in G.E$
4	RELAX (u, v, w)
5	**for** each edge $(u, v) \in G.E$
6	**if** $v.d > u.d + w(u, v)$
7	**return** $FALSE$
8	**return** $TRUE$

Iteration	RELAX(u, v)	$t.d$	$x.d$	$y.d$	$z.d$
$i = 1$	(s, t)	6			
	(s, y)			7	
	(t, x)		11		
	(t, y)				
	(t, z)			2	

BELLMAN-FORD ALGORITHM
IN ACTION

- Apply the BELLMAN-FORD algorithm on the graph.



Adjacency list

$s \to t \to y$

$t \to x \to y \to z$

$x \to \boldsymbol{t}$

$y \to x \to z$

$z \to s \to x$

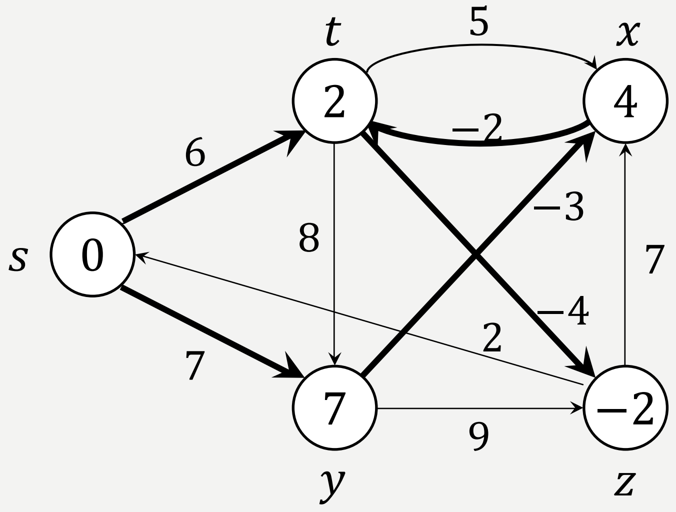
BELLMAN-FORD $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$

```
INITIALIZE-SINGLE-SOURCE (G, s)
for i = 1 to |G.V| − 1
        for each edge (u, v) ∈ G.E
                RELAX (u, v, w)
for each edge (u, v) ∈ G.E
        if v.d > u.d + w(u, v)
                return FALSE
return TRUE
```

Iteration	RELAX(u,v)	$t.d$	$x.d$	$y.d$	$z.d$
$i = 1$	(s,t)	6			
	(s,y)			7	
	(t,x)		11		
	(t,y)				
	(t,z)			2	
	(x,t)				

BELLMAN-FORD ALGORITHM
STATE AFTER LINE 2 ~ 4

- Apply the BELLMAN-FORD algorithm on the graph.



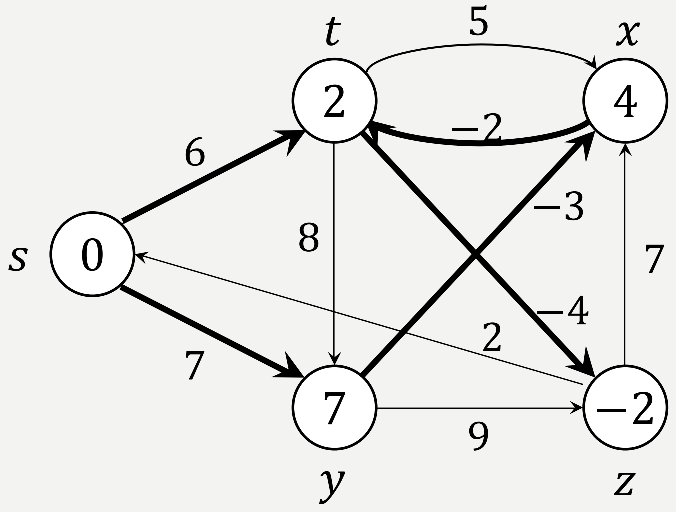
Adjacency list
$s \to t \to y$
$t \to x \to y \to z$
$x \to \boldsymbol{t}$
$y \to x \to z$
$z \to s \to x$

	BELLMAN-FORD $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$
1	INITIALIZE-SINGLE-SOURCE (G, s)
2	**for** $i = 1$ **to** $\lvert G.V \rvert - 1$
3	**for** each edge $(u, v) \in G.E$
4	RELAX (u, v, w)
5	**for** each edge $(u, v) \in G.E$
6	**if** $v.d > u.d + w(u, v)$
7	**return** $FALSE$
8	**return** $TRUE$

Iteration	RELAX(u, v)	$t.d$	$x.d$	$y.d$	$z.d$
$i = 1$	(s, t)	6			
	(s, y)			7	
	(t, x)		11		
	(t, y)				
	(t, z)			2	
	(x, t)				

BELLMAN-FORD ALGORITHM
LINE 5 ~ 7

- Apply the BELLMAN-FORD algorithm on the graph.



Adjacency list

$s \to t \to y$

$t \to x \to y \to z$

$x \to t$

$y \to x \to z$

$z \to s \to x$

BELLMAN-FORD $(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$	
1	INITIALIZE-SINGLE-SOURCE (G, s)
2	**for** $i = 1$ **to** $\lvert G.V \rvert - 1$
3	**for** each edge $(u, v) \in G.E$
4	RELAX (u, v, w)
5	**for** each edge $(u, v) \in G.E$
6	**if** $v.d > u.d + w(u, v)$
7	**return** $FALSE$
8	**return** $TRUE$

BELLMAN-FORD ALGORITHM
RUNNING TIME

- Initialization costs $\Theta($ _____ $)$.
- Line 2 through 4 can be abstracted as a doubly-nest **for**-loop.
  - Outer **for** of the doubly-nested loop runs in $\Theta($ _____ $)$.
  - Inner **for** of the doubly-nested loop runs $\Theta($ _____ $)$.
- Line 5 through 7 is a **for** loop that runs $\Theta($ _____ $)$.
- Overall running time $T = O($ _____ $)$.

BELLMAN-FORD $(\boldsymbol{G, w, s})$	
1	INITIALIZE-SINGLE-SOURCE (G, s)
2	**for** $i = 1$ **to** $\lvert G.V \rvert - 1$
3	**for** each edge $(u, v) \in G.E$
4	RELAX (u, v, w)
5	**for** each edge $(u, v) \in G.E$
6	**if** $v.d > u.d + w(u, v)$
7	**return** $FALSE$
8	**return** $TRUE$

NEXT UP WRAPPING UP

- Touch on MST
- P, NP, NP-complete

REFERENCE

- Screenshots are taken from the textbook.