COMP 251

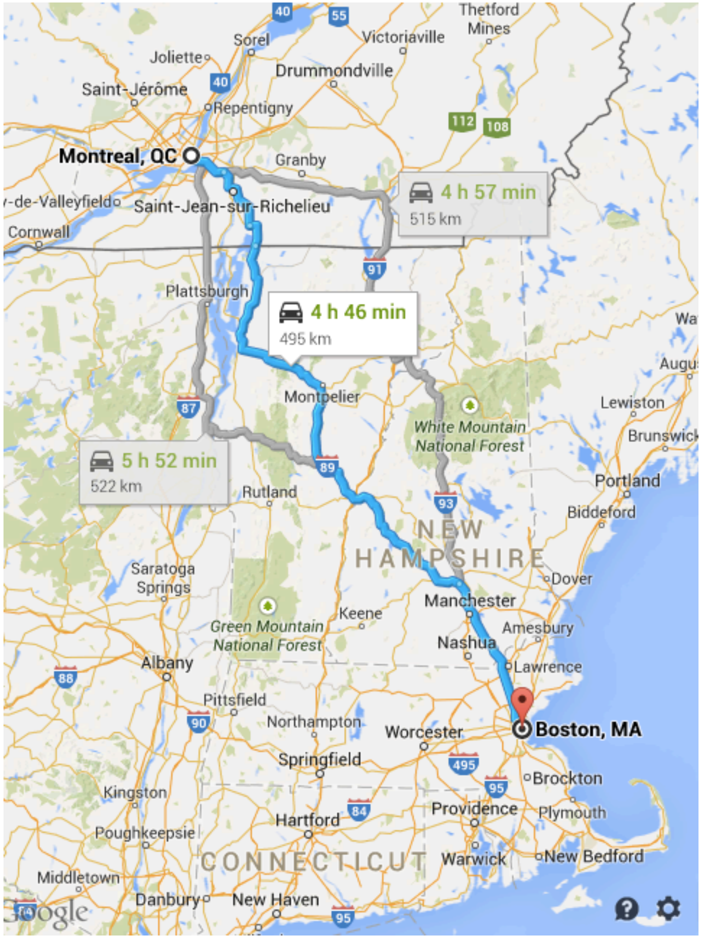
Algorithms & Data Structures (Winter 2021)

Graphs – Single Source Shortest Paths

School of Computer Science
McGill University

Slides of (Comp321 ,2021), Langer (2014), Kleinberg & Tardos, 2005 & Cormen et al., 2009, Jaehyun Park' slides CS 97SI, Top-coder tutorials, T-414-AFLV Course, Programming Challenges books, slides from D. Plaisted (UNC) and Comp251-Fall McGill.

Shortest Path - Problem



What is the shortest road to go from one city to another?

Example: Which road should I take to go from Montréal to Boston (MA)?

Variants:

- What is the fastest road?
- What is the cheapest road?

Shortest Path – As a graph problem

Input:

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

Weight of path $p = \langle v_0, v_1, \ldots, v_k \rangle$

$$= \sum_{k=1}^{n} w(v_{k-1}, v_k)$$

= sum of edges weights on path *p*

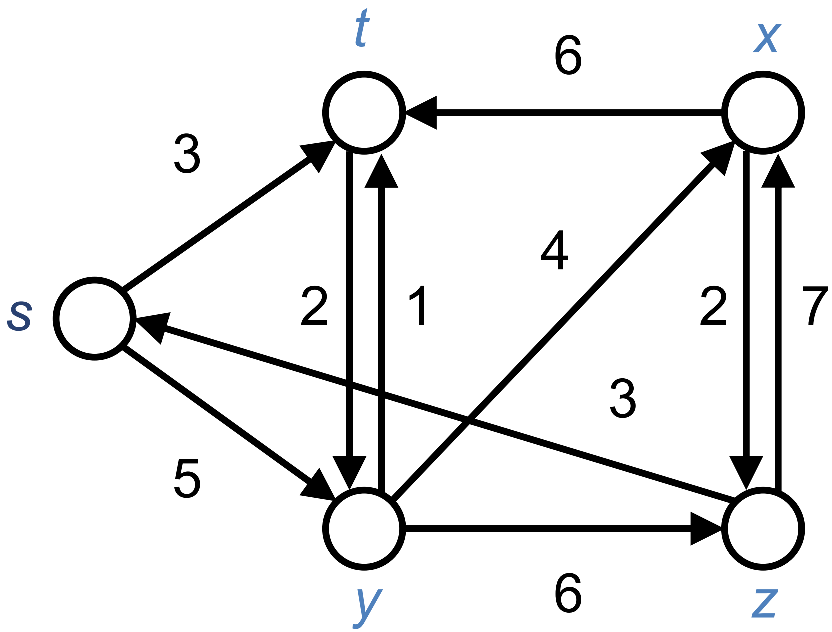
Shortest-path weight *u* to *v*:

$$\delta(u,v) = \begin{cases} \min\left\{ w(p) : u \overset{p}{\mapsto} v \right\} & \text{If there exists a path } u \rightsquigarrow v. \\ \infty & \text{Otherwise.} \end{cases}$$

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

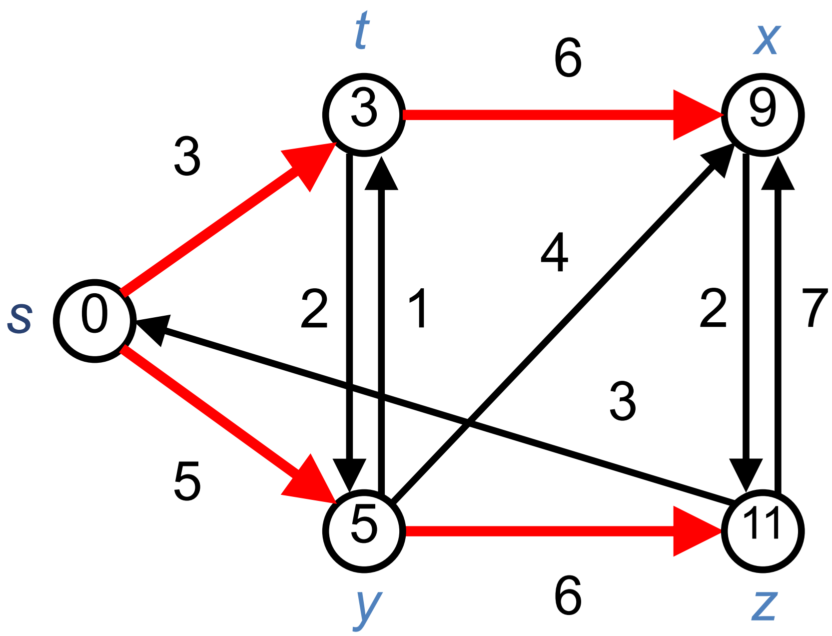
Generalization of breadth-first search to weighted graphs

Shortest Path – As a graph problem - example



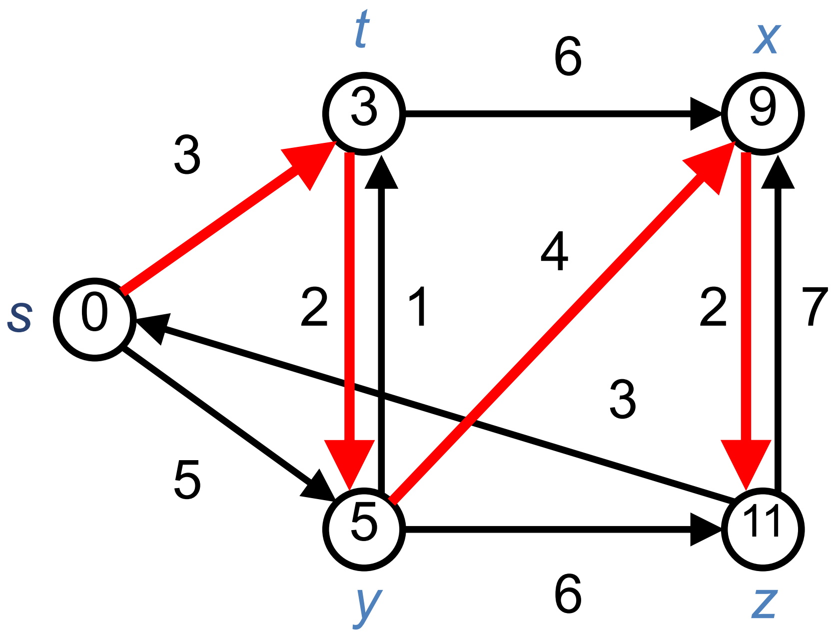
Shortest path from s?

Shortest Path – As a graph problem - example



Shortest paths are organized as a tree.
Vertices store the length of the shortest path from s.

Shortest Path – As a graph problem - example



Shortest paths are not necessarily unique!

Shortest Path – As a graph problem - variants

- **Single-source (SSSP):** Find shortest paths from a given source vertex $s \in V$ to every vertex $v \in V$.
- **Single-destination:** Find shortest paths to a given destination vertex.
  - By reversing the direction of each edge in the graph, you reduce it to SSSP.
- **Single-pair:** Find shortest path from u to v.
  - SSSP solves this variant. All known algorithms for this problem have the same worst-case asymptotic running time as the best single-source algorithm.
- **All-pairs:** Find shortest path from u to v for all $u, v \in V$.
  - By running a SSSP algorithm once from each vertex, but we can solve it faster.

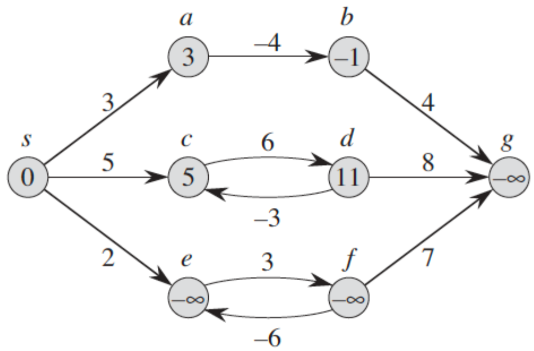
Shortest Path – Negative weight edges

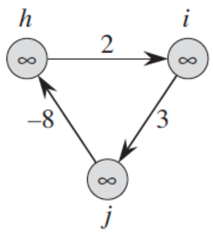
Negative weight edges can create issues.

Why? If we have a negative-weight cycle, we can just keep going around it, and get *w(s, v)* = −∞ for all v on the cycle.

When? If they are reachable from the source (Corollary: It is OK to have a negative-weight cycles if it is not reachable from the source).

What? Some algorithms work only if there are no negative-weight edges in the graph. We must specify when they are allowed and not.





Shortest Path – Cycles

Shortest paths cannot contain cycles:

- Negative-weight: Already ruled out.
- Positive-weight: we can get a shorter path by omitting the cycle.
- Zero-weight: no reason to use them ⇒ assume that our solutions will not use them.

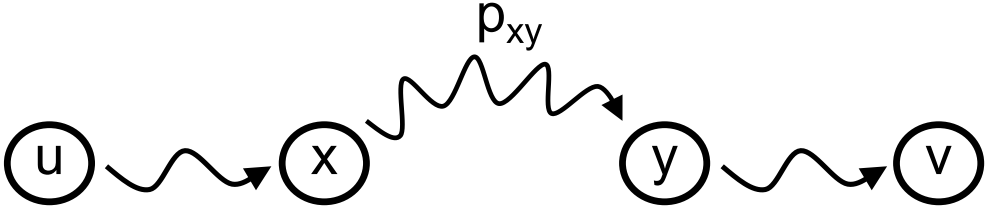
Consequence:

- Since any acyclic path in a graph $G = (V,E)$ contains at most $|V|$ distinct vertices, it also contains at most $|V| - 1$ edges. Thus, we can restrict our attention to shortest paths of at most $|V| - 1$ edges.

Shortest Path – Optimal substructure

Lemma
Any subpath of a shortest path is a shortest path.

Proof: (cut and paste)



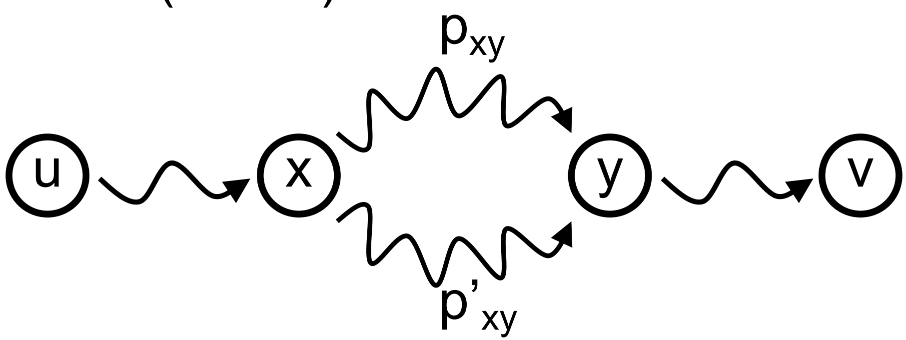
Suppose this path *p* is a shortest path from *u* to *v*.

Then $\delta(u,v) = w(p) = w(p_{ux}) + w(p_{xy}) + w(p_{yv})$.

Shortest Path – Optimal substructure

> ***Lemma***
> Any subpath of a shortest path is a shortest path.

Proof: (cont'd)



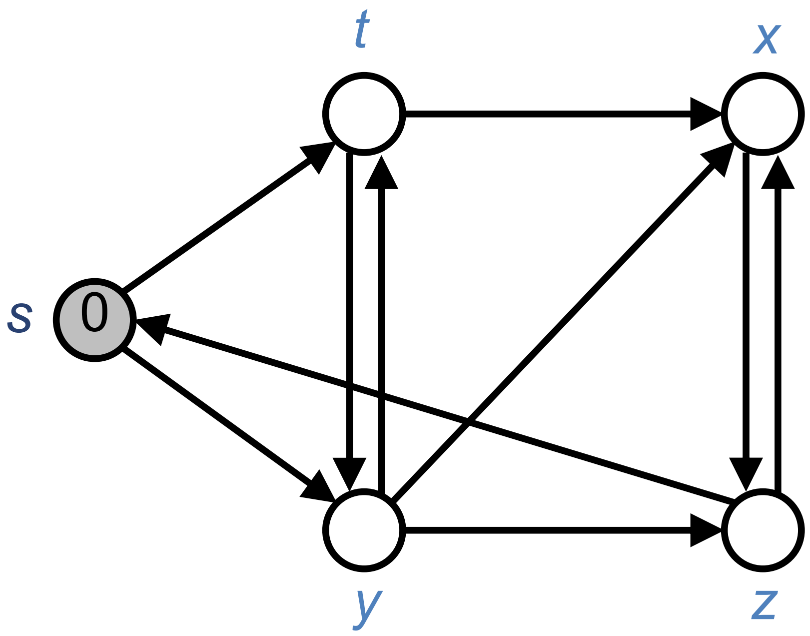
Now suppose there exists a shorter path $x \overset{p'_{xy}}{\rightsquigarrow} y$.

Then $w(p'_{xy})<w(p_{xy})$.

$w(p') = w(p_{ux}) + w(p'_{xy}) + w(p_{yv}) < w(p_{ux}) + w(p_{xy}) + w(p_{yv}) = w(p)$.

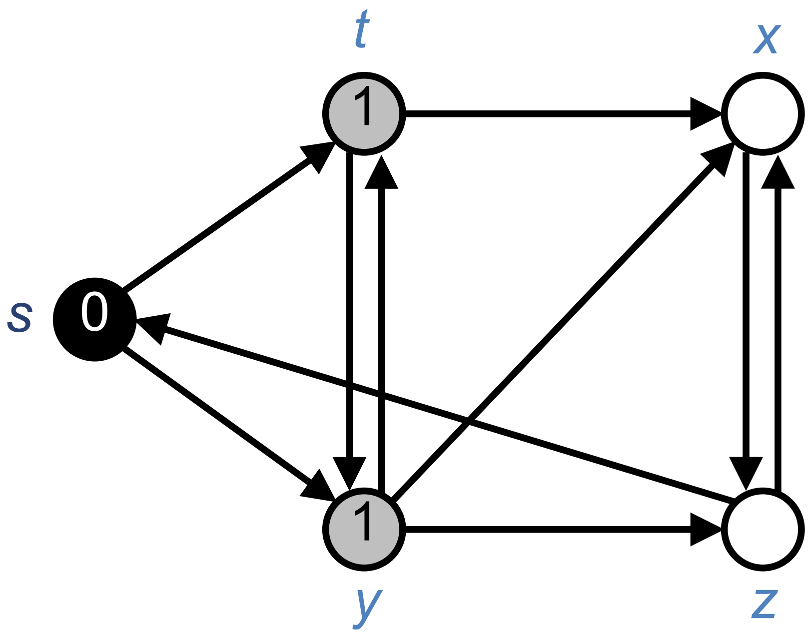
Contradiction of the hypothesis that p is the shortest path!

Shortest Path – Customized BFS

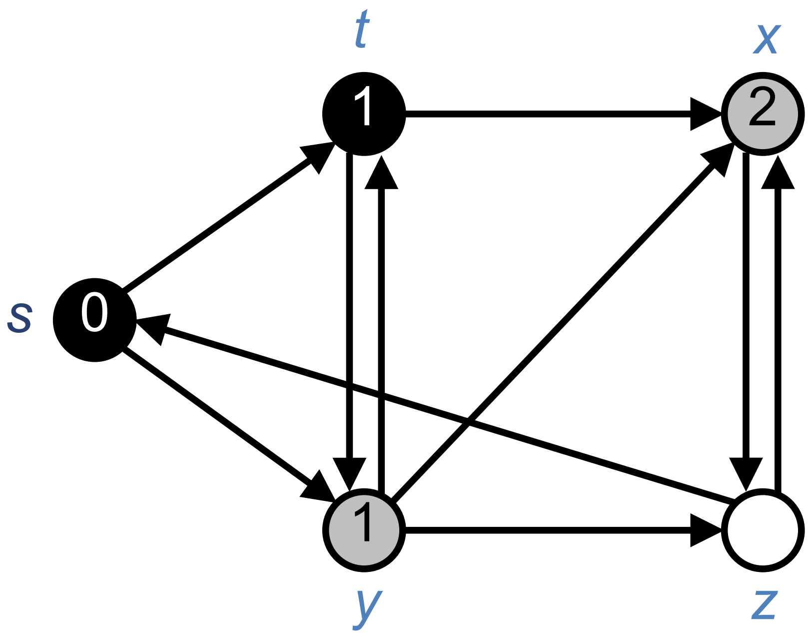


Vertices count the number of edges used to reach them.

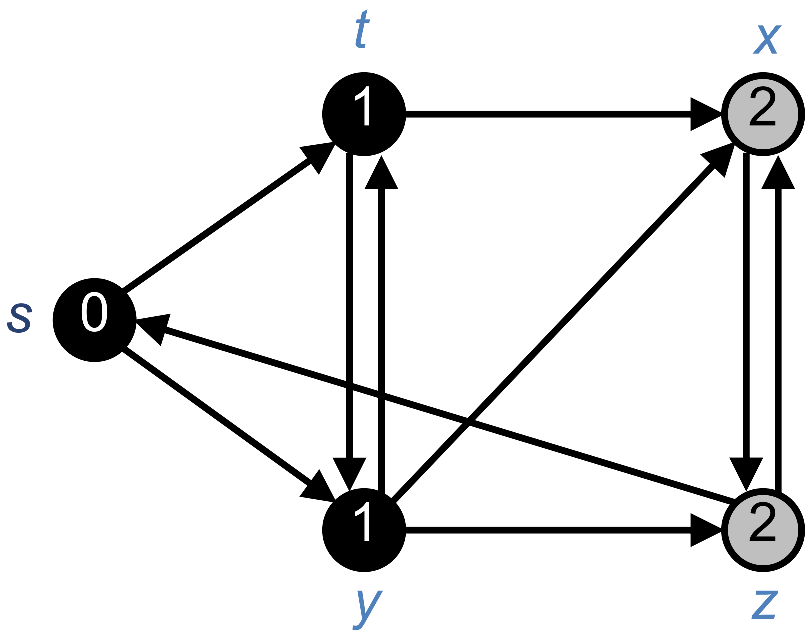
Shortest Path – Customized BFS



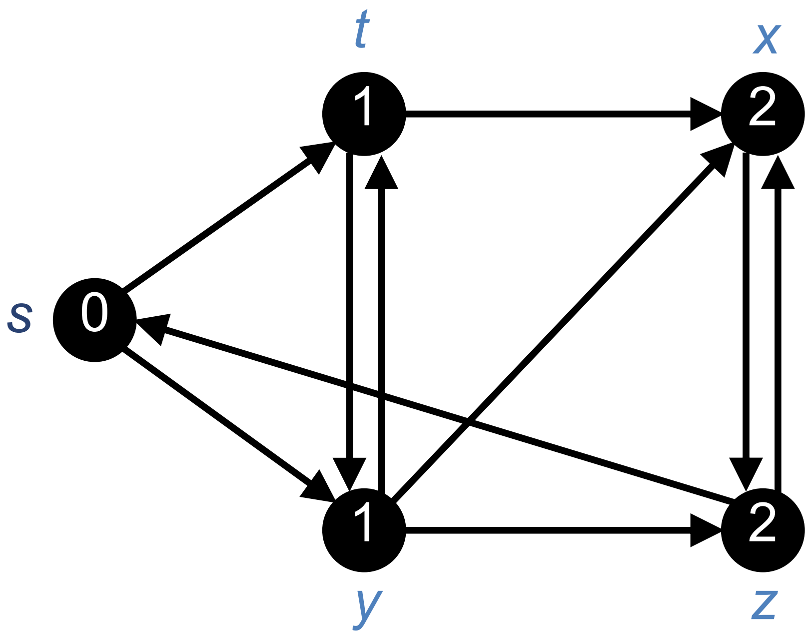
Shortest Path – Customized BFS



Shortest Path – Customized BFS



Shortest Path – Customized BFS



Can we generalize BFS to use edge weights?

SSSP – Output

For each vertex $v \in V$:

- $d[v] = \delta(s,v)$.
  - Initially, $d[v] = \infty$.
  - Reduces as algorithms progress, but always maintain $d[v] \geq \delta(s,v)$.
  - Call $d[v]$ a **shortest-path estimate**.
- $\pi[v]$ = predecessor of v on a shortest path from *s*.
  - If no predecessor, $\pi[v]$ = NIL.
  - π induces a tree - **shortest-path tree** (see proof in textbook).

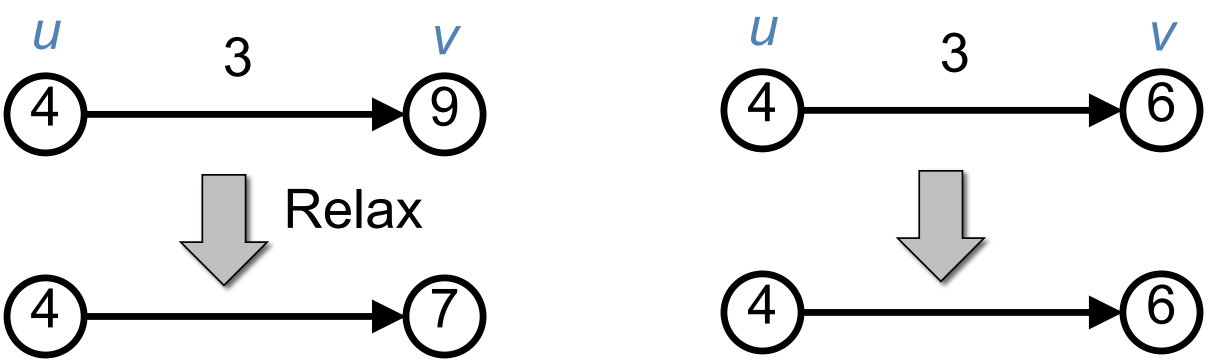
SSSP – Structure

1. Initialization

```
INIT-SINGLE-SOURCE(V,s)
  for each v ∈ V do
    d[v]← ∞
    π[v]← NIL
  d[s]←0
```

SSSP – Structure

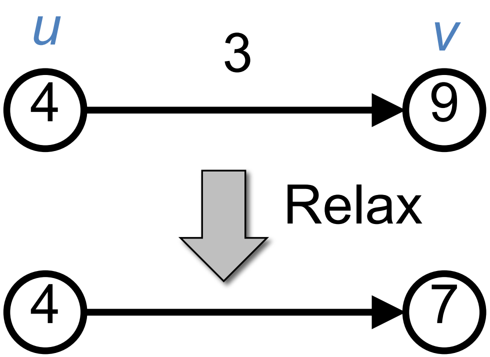
2. Scan vertices and relax edges.
   - An edge u -> v is tense if d(u)+w(*u*,v)< d(v).
   - If u->v is tense, the tentative shortest path s $\rightsquigarrow$ v is clearly incorrect, because the path s $\rightsquigarrow$ u->v is shorter.

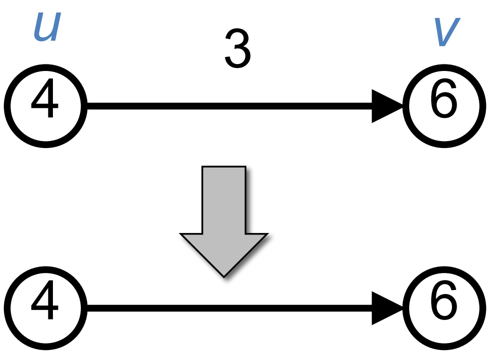


SSSP – Structure

2. Scan vertices and relax edges.

```
RELAX(u,v,w)
   if d[v]>d[u]+w(u,v) then
      d[v] ← d[u]+w(u,v)
      π[v]←u
```





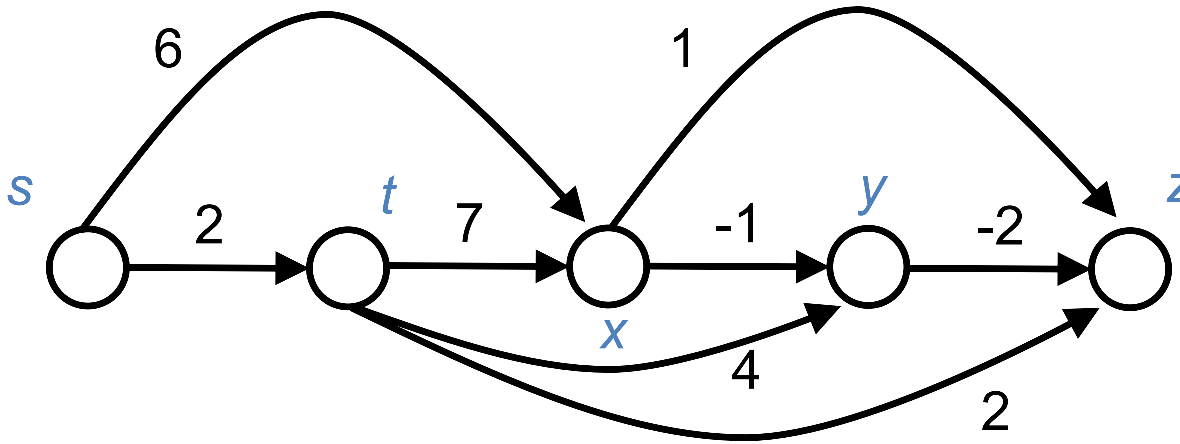
Shortest Path and Relaxation – Properties

- Triangle inequality
  - For any edge $(u, v) \in E$, we have $\delta(s, v) \leq \delta(s, u) + w(u, v)$.
- Upper-bound property
  - We always have $\mathrm{d[v]} \geq \delta(s, v)$ for all vertices $v \in V$, and once $\mathrm{d[v]}$ achieves the value $\delta(s, v)$, it never changes.
- No-path property
  - If there is no path from *s* to *v*, then we always have $\mathrm{d[v]} = \delta(s, v) = \infty$.
- Convergence property.
  - If s $\rightsquigarrow$ u -> v is a shortest path in *G* for some $u, v \in V$, and if $d[u] = \delta(s, u)$ at any time prior to relaxing edge (u,v), then $d[v] = \delta(s, v)$ at all time afterwards.
- Path-relaxation property
  - If $p = \langle v_0, v_i, \dots, v_k \rangle$ is a shortest path from $s = v_0$ to v_k, and we relax the edges of *p* in the order $(v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k)$ then $v_k[d] = \delta(s, v_k)$
- Predecessor-subgraph property
  - Once $v[d] = \delta(s, v)$ for all $v \in V$, the predecessor subgraph is a shortest-paths tree rooted at s.

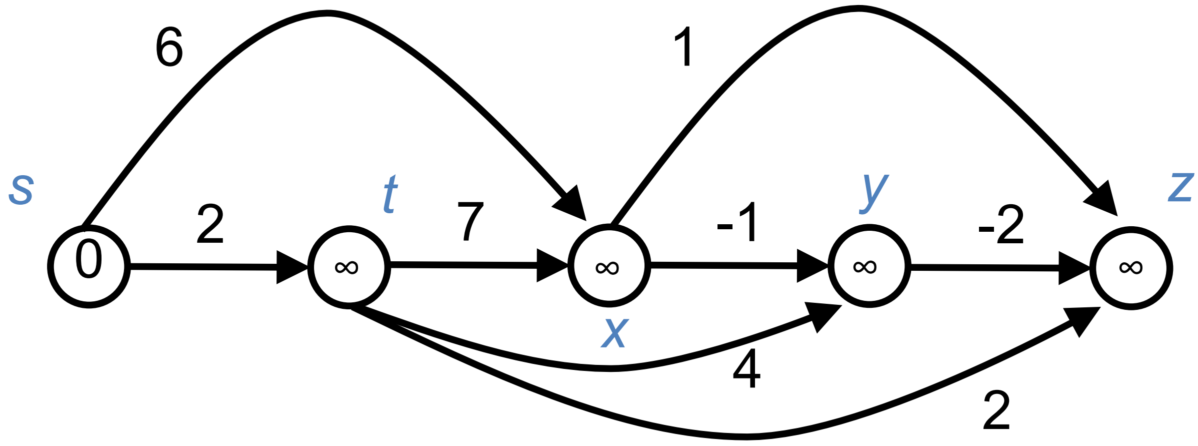
SSSP – DAG

DAG ⇒ no negative-weight cycles.

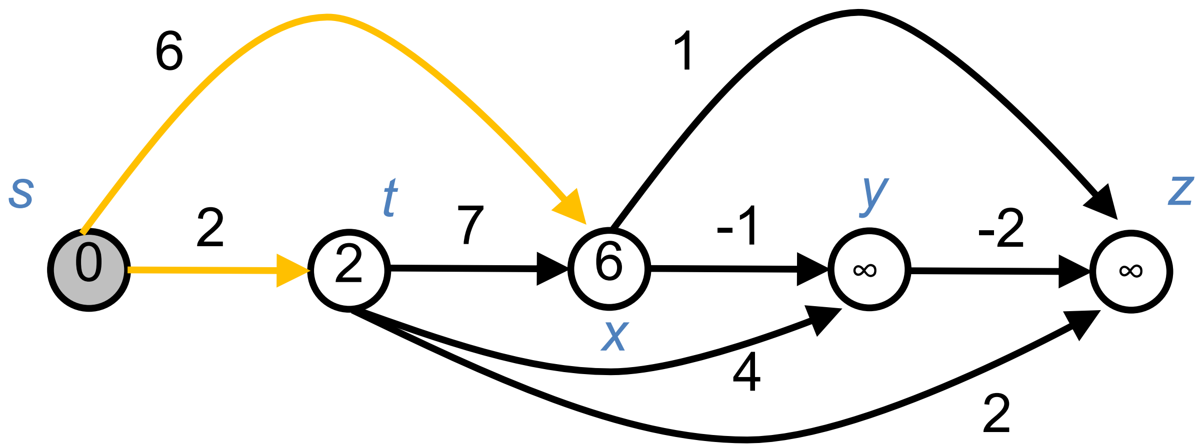
```
DAG-SHORTEST-PATHS(V,E,w,s)
topologically sort the vertices
INIT-SINGLE-SOURCE(V,s)
for each vertex u in topological order do
   for each vertex v∈Adj[u] do
      RELAX(u,v,w)
```



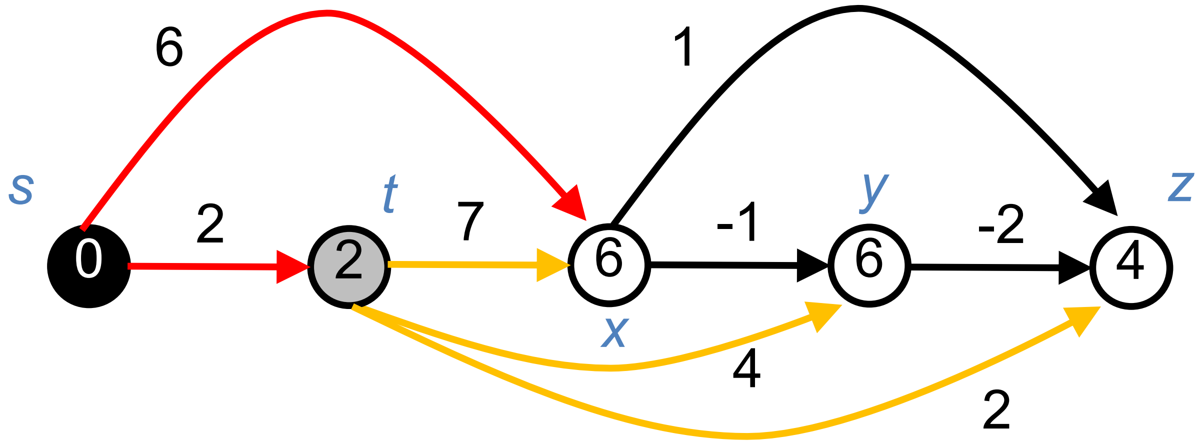
SSSP – DAG - Example



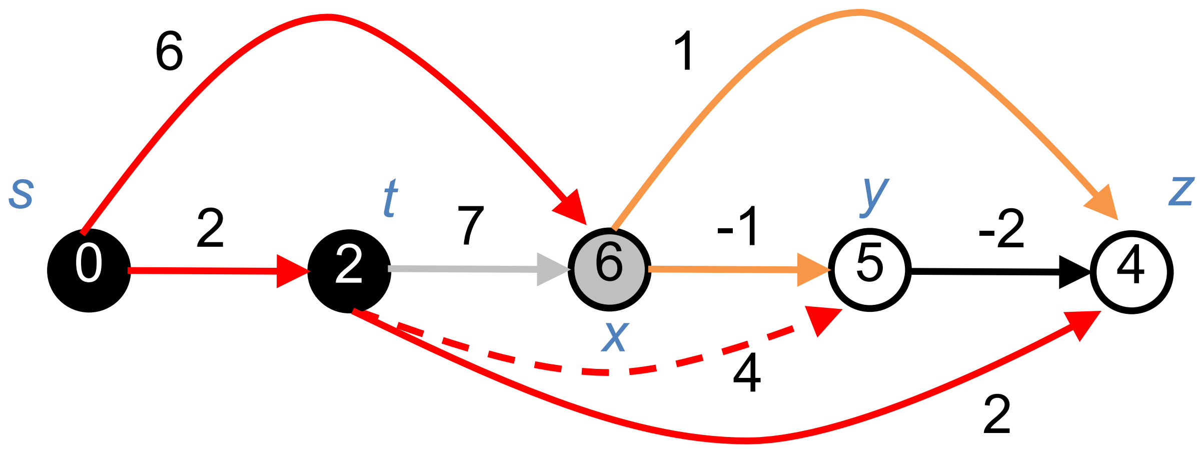
SSSP – DAG - Example



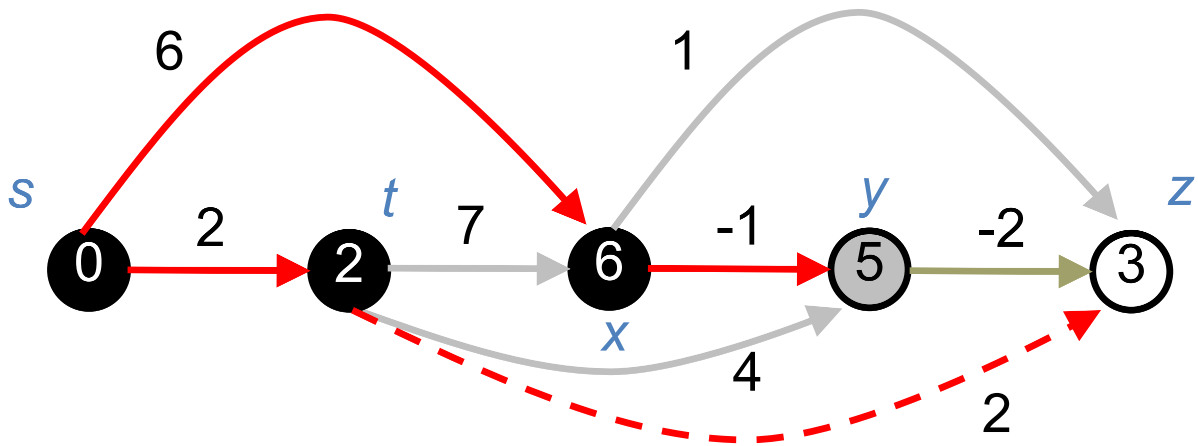
SSSP – DAG - Example



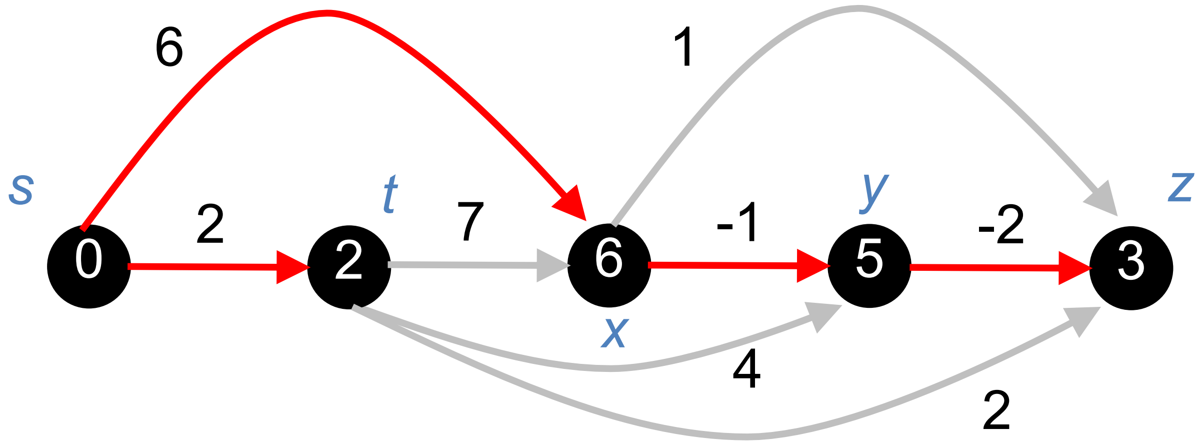
SSSP – DAG - Example



SSSP – DAG - Example



SSSP – DAG - Example



SSSP – DAG

```
DAG-SHORTEST-PATHS(V,E,w,s)
topologically sort the vertices          O(V+E)
INIT-SINGLE-SOURCE(V,s)                  O(V)
for each vertex u in topological order do
   for each vertex v ∈ Adj[u] do           O(V+E)
      RELAX(u,v,w)
```

Time: (V + E).

Correctness:

Because we process vertices in topologically sorted order, edges of **any** path must be relaxed in order of appearance in the path. ⇒ Edges on any shortest path are relaxed in order.

⇒ By path-relaxation property, correct.

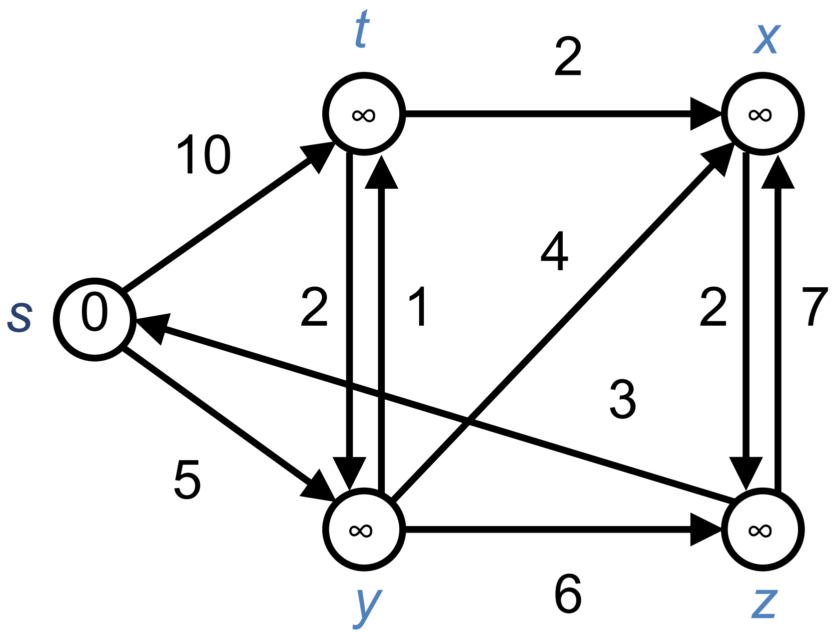
SSSP – Dijkstra's algorithm

- No negative-weight edges.
- Weighted version of BFS:
  - **Instead of a FIFO queue, uses a priority queue**.
  - Keys are shortest-path weights (d[v]).
- Have two sets of vertices:
  - S = vertices whose final shortest-path weights are determined,
  - Q = priority queue = $V - S$.
- Greedy choice: At each step we choose the light edge.

SSSP – Dijkstra's algorithm

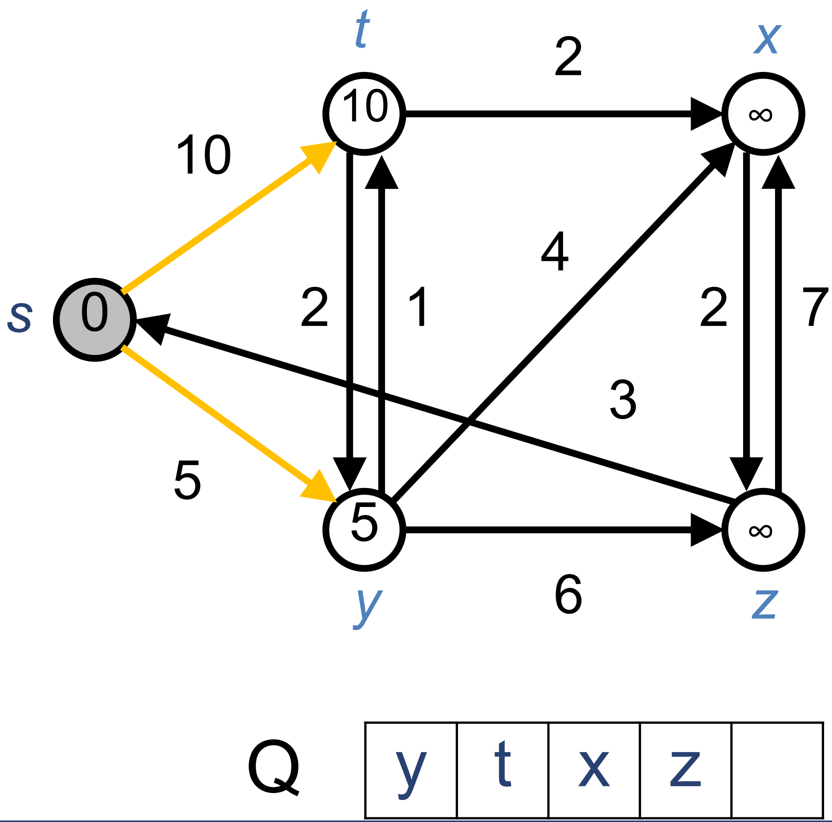
```
DIJKSTRA(V,E,w,s)
INIT-SINGLE-SOURCE(V,s)
S ← ∅
Q ← V
while Q ≠ ∅ do
   u ← EXTRACT-MIN(Q)
   S ← S ∪ {u}
   for each vertex v ∈ Adj[u] do
      RELAX(u,v,w)
```

SSSP – Dijkstra's algorithm - Example

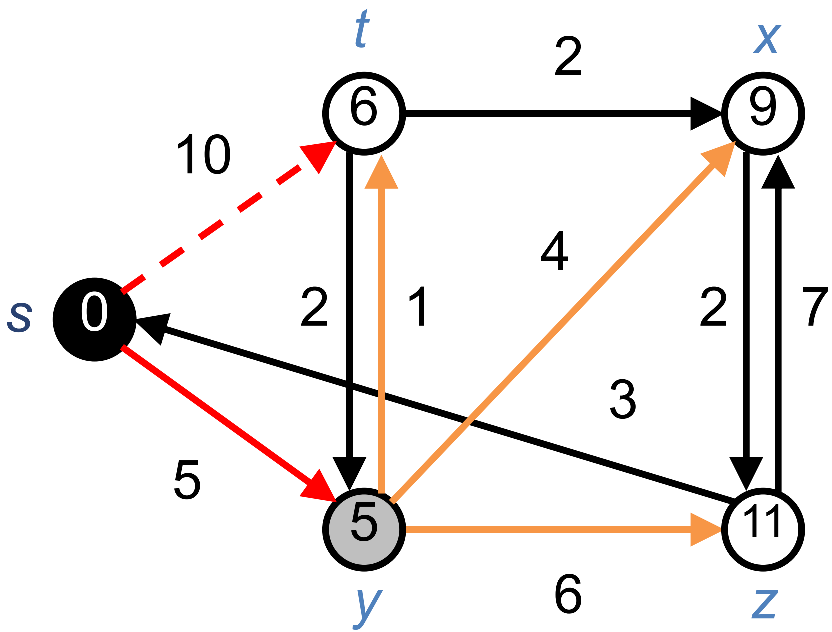


Q	s	t	y	x	z

SSSP – Dijkstra's algorithm - Example

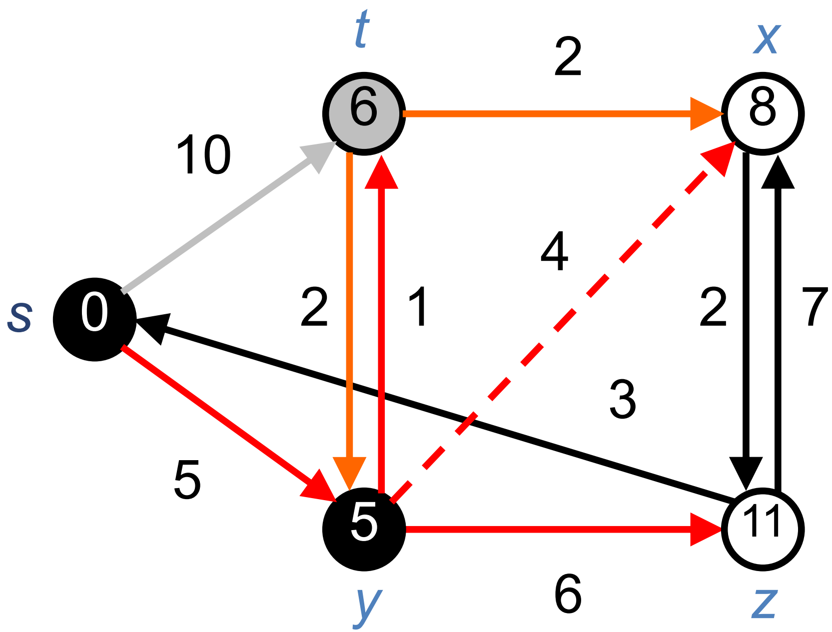


SSSP – Dijkstra's algorithm - Example



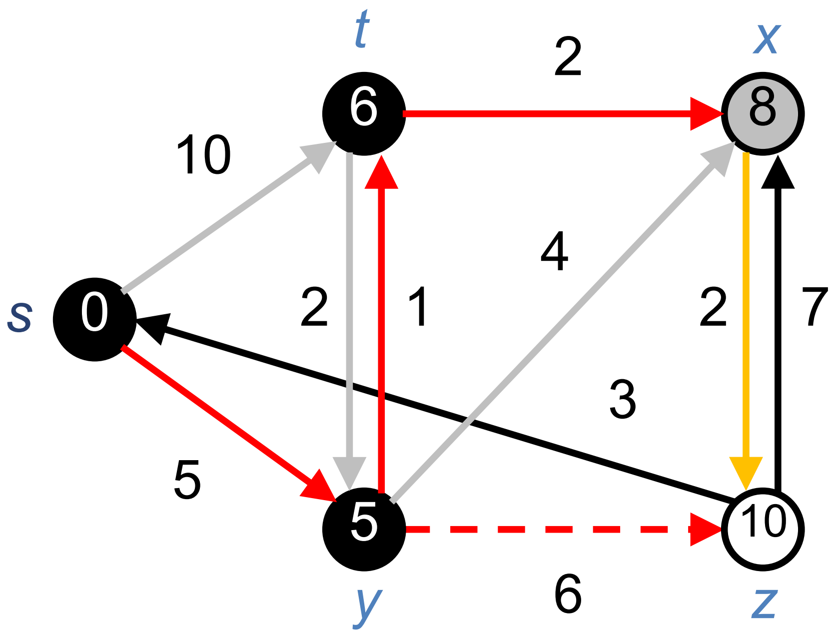
Q					
	t	x	z		

SSSP – Dijkstra’s algorithm - Example



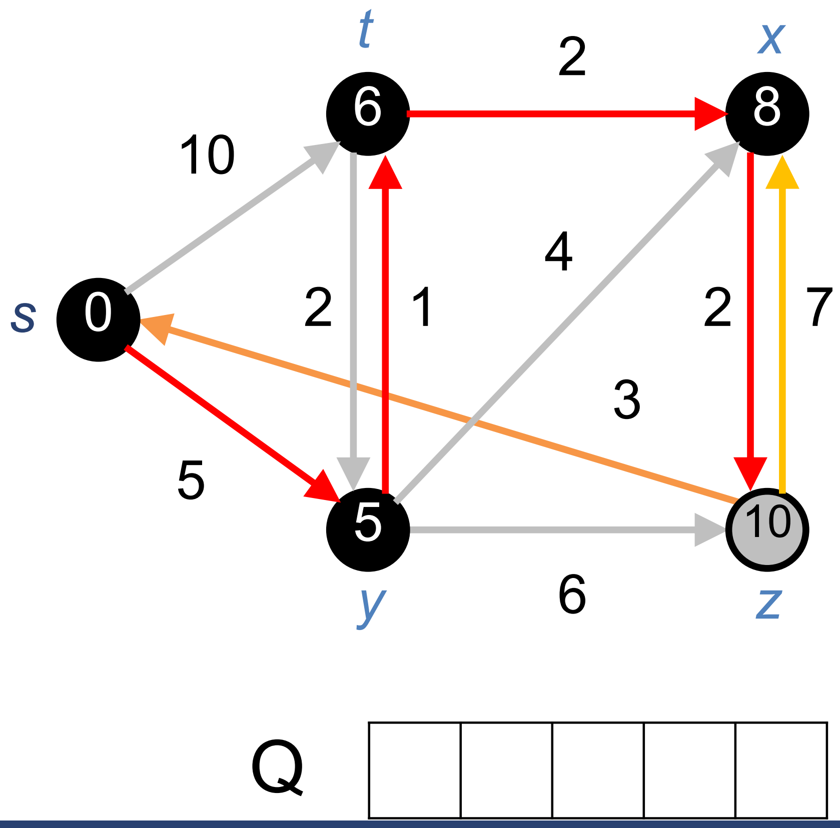
Q	x	z			

SSSP – Dijkstra's algorithm - Example

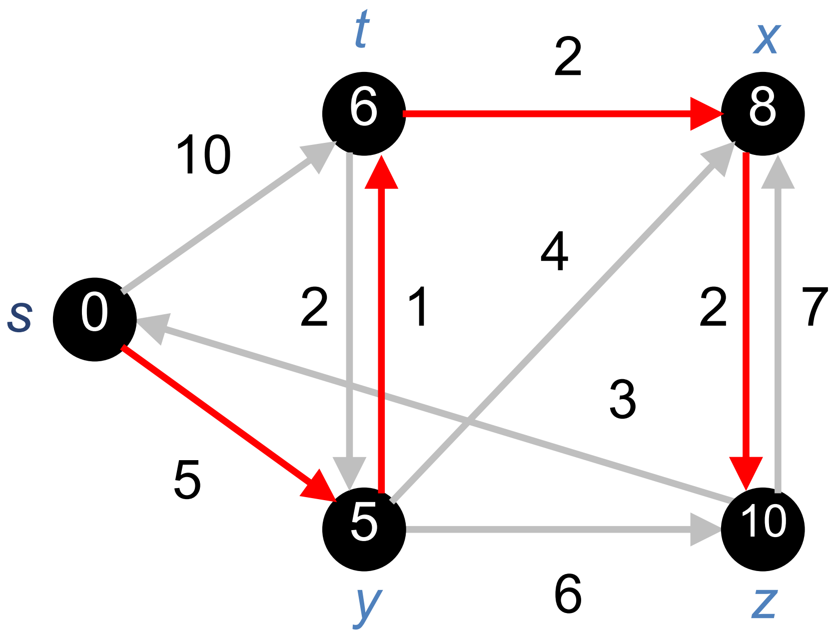


Q					
	z				

SSSP – Dijkstra's algorithm - Example



SSSP – Dijkstra's algorithm - Example



Dijkstra's algorithm - Correctness

Loop invariant:
At the start of each iteration of the while loop, $d[v] = \delta(s,v)$ for all $v \in S$.

Initialization:
Initially, $S = \emptyset$, so trivially true.

Termination:
At end, $Q=\emptyset \Rightarrow S = V \Rightarrow d[v] = \delta(s,v)$ for all $v \in V$.

Maintenance:
Show that $d[u] = \delta(s,u)$ when u is added to S in each Iteration.

Dijkstra's algorithm - Correctness

Show that $d[u] = \delta(s,u)$ when u is added to S in each iteration.

Suppose there exists u such that $d[u] \neq \delta(s,u)$.

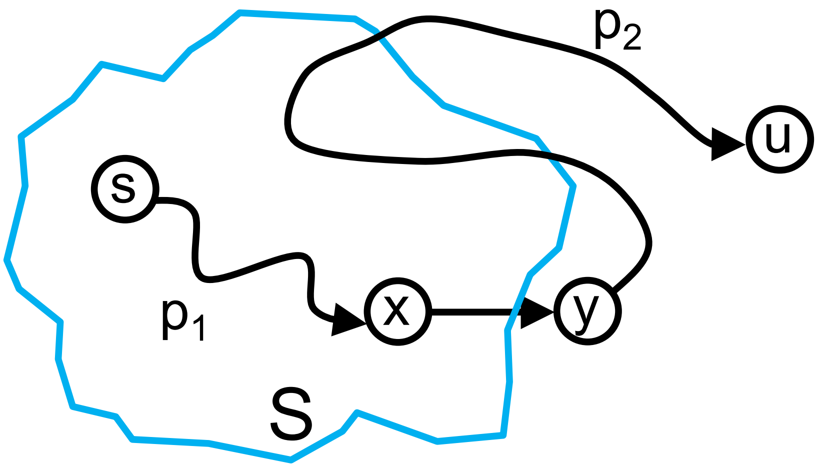
Let u be the first vertex for which $d[u] \neq \delta(s, u)$ when u is added to S.

- $u \neq s$, since $d[s] = \delta(s,s) = 0$.
- Therefore, $s \in S$, so $S \neq \emptyset$.
- There must be some path $s \leadsto u$. Otherwise $d[u] = \delta(s,u) = \infty$ by no-path property.
- So, there is a path $s \leadsto u$. Thus, there is **a shortest *p* path $\boldsymbol{s \leadsto u}$**.

Dijkstra's algorithm - Correctness

Show that $d[u] = \delta(s,u)$ when u is added to S in each iteration.

Just before u is added to S, the path p connects a vertex in S (i.e., s) to a vertex in $V - S$ (i.e., u).
Let y be the first vertex along p that is in $V - S$ and let $x \in S$ be the predecessor of y.



Decompose p into $s \overset{p_1}{\rightsquigarrow} x \rightarrow y \overset{p_2}{\rightsquigarrow} u$.

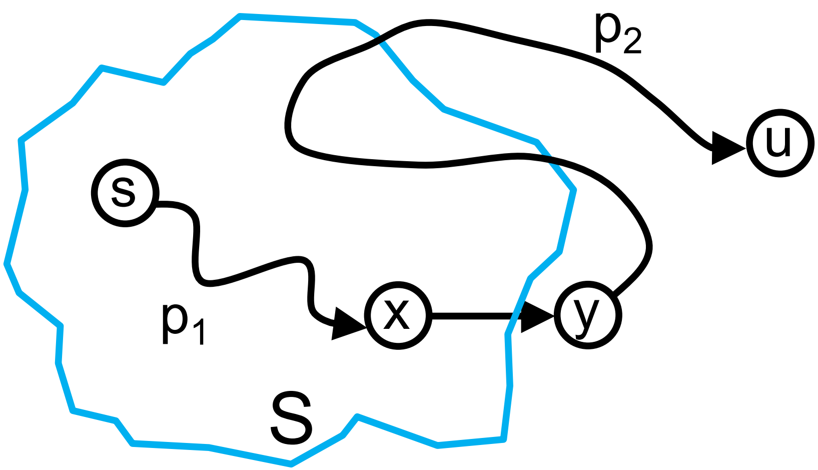
Dijkstra's algorithm - Correctness

Claim 1: d[y] = δ(s, y) when u is added to S.

Proof:

x ∈ S and u is the first vertex such that d[u] ≠ δ(s, u) when u is added to S ⇒ d[x] = δ(s, x) when x is added to S.

But when x is added we relax the edge (x, y), so by the *convergence property*, d[y] = δ(s, y).



Dijkstra's algorithm - Correctness

Show that $d[u] = \delta(s,u)$ when u is added to S in each iteration.

Now, we can get a contradiction to $d[u] \neq \delta(s, u)$:

y is on shortest path p ($s \rightsquigarrow u$), and all edge weights are nonnegative.

$\Rightarrow \delta(s, y) \leq \delta(s, u)$

Then by Claim 1, we have $d[y] = \delta(s,y)$

$\leq \delta(s,u)$

$\leq d[u]$ (upper-bound property)

In addition, ***y* and *u* were in *Q* when we chose *u*, thus *d*[*u*] ≤ *d*[*y*].**

We have $d[y] \leq d[u]$ & $d[u] \leq d[y] \Rightarrow d[u] = d[y]$.

Therefore, $d[y] = \delta(s, y) \leq \delta(s, u) \leq d[u] = d[y]$

Contradicts assumption that $d[u] \neq \delta(s,u)$. ■

Dijkstra's algorithm - Analysis

DIJKSTRA(V,E,w,s)	
INIT-SINGLE-SOURCE(V,s)	
$S \leftarrow \emptyset$	
$Q \leftarrow V$	INSERT
while $Q \neq \emptyset$ **do**	O(V)
$u \leftarrow$ EXTRACT-MIN(Q)	EXTRACT_MIN
$S \leftarrow S \cup \{u\}$	
for each vertex v $\in$ $Adj[u]$ **do**	O(E)
RELAX(u,v,w)	DECREASE-KEY

Dijkstra’s algorithm - Analysis

It depends on implementation of priority queue.

If binary heap, each operation takes $O(\lg V)$ time
$\Rightarrow O(E \lg V)$.

Note: We can achieve $O(V \lg V + E)$ with Fibonacci heaps.

Outline

- Graphs.
  - Introduction.
  - Topological Sort. / Strong Connected Components
  - Network Flow 1.
    - Introduction
    - Ford-Fulkerson
  - Network Flow 2.
    - Min-cuts
  - Shortest Path.
  - Minimum Spanning Trees.
  - Bipartite Graphs.


time for questions