COMP251: Single source shortest paths

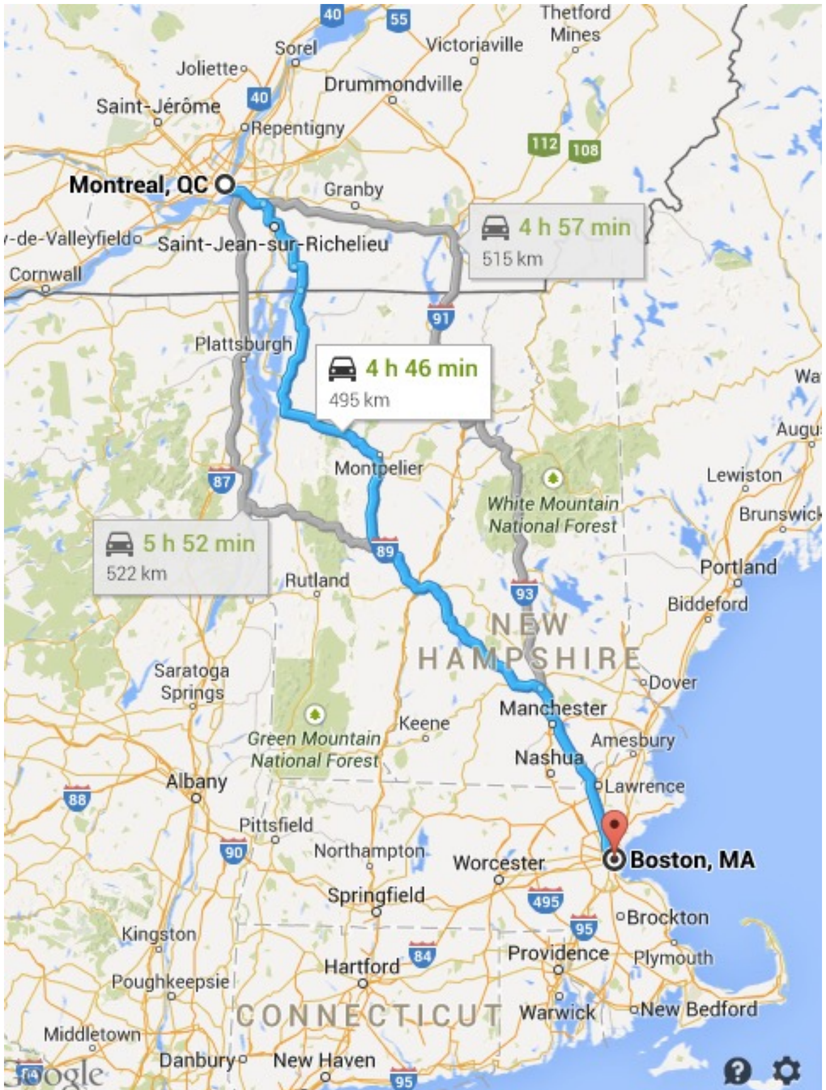
Jérôme Waldispühl & Roman Sarrazin-Gendron
School of Computer Science
McGill University

Based on (Cormen *et al.*, 2002)

Outline

- Introduction
- Optimal Substructure
- Definitions & properties
  - Edge relaxation
  - Triangle inequality
  - Upper bound property
  - No-path property
  - Convergence property
  - Path relaxation property
- Single source shortest path in DAGs
- Dijkstra's Algorithm

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?

Modeling as graphs

Input:

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

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

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

= sum of edges weights on path *p*

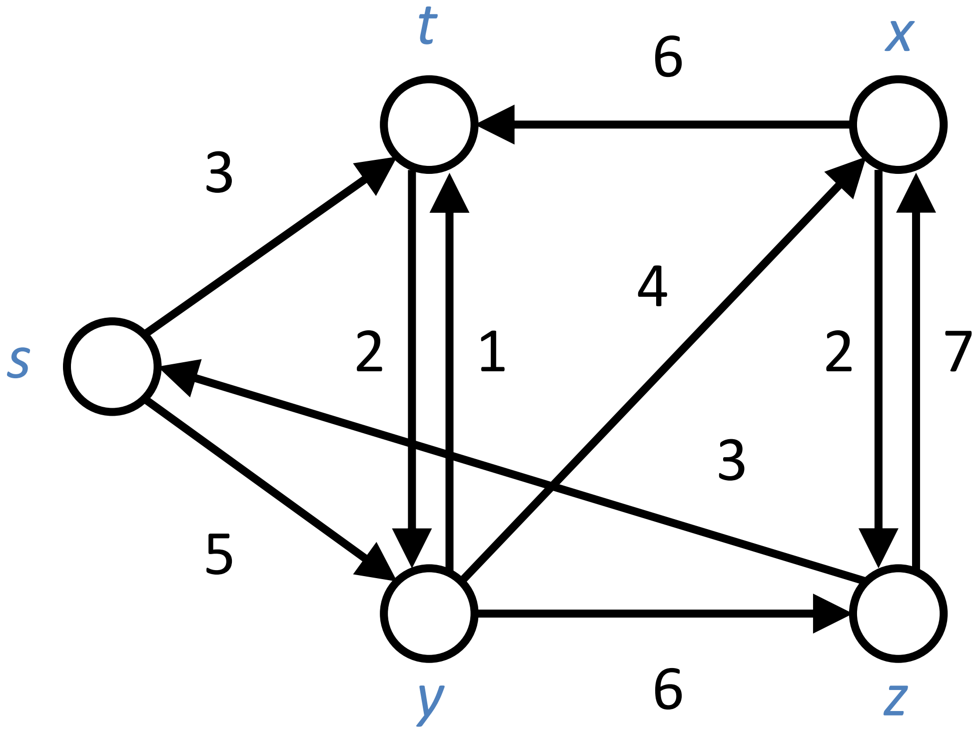
Shortest-path weight *u* to *v*:

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

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

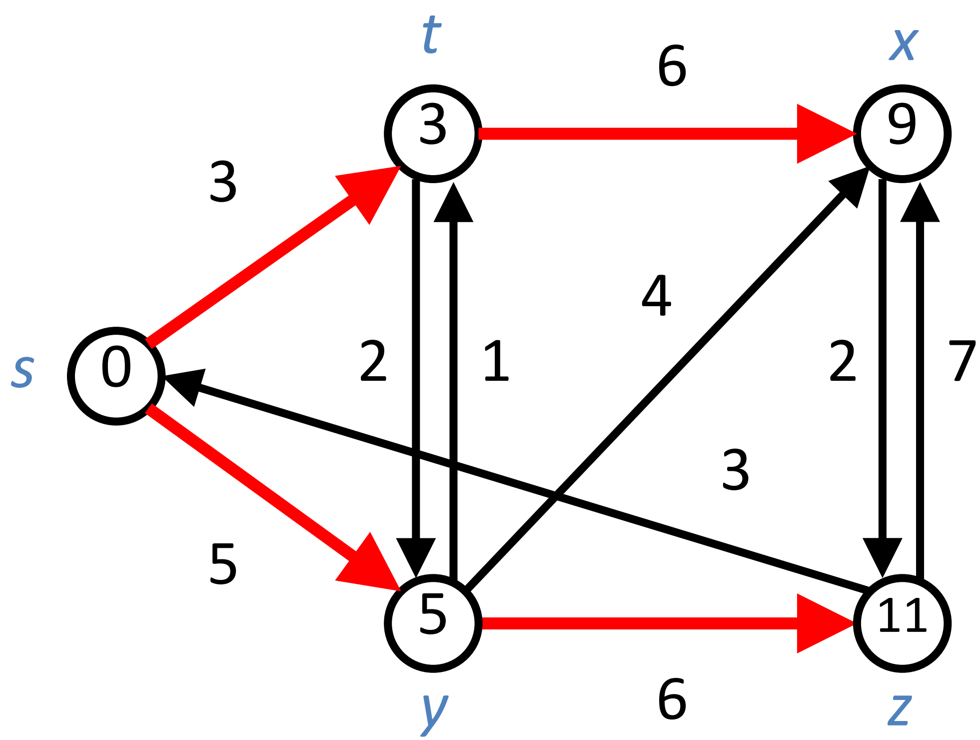
Generalization of breadth-first search to weighted graphs

Example



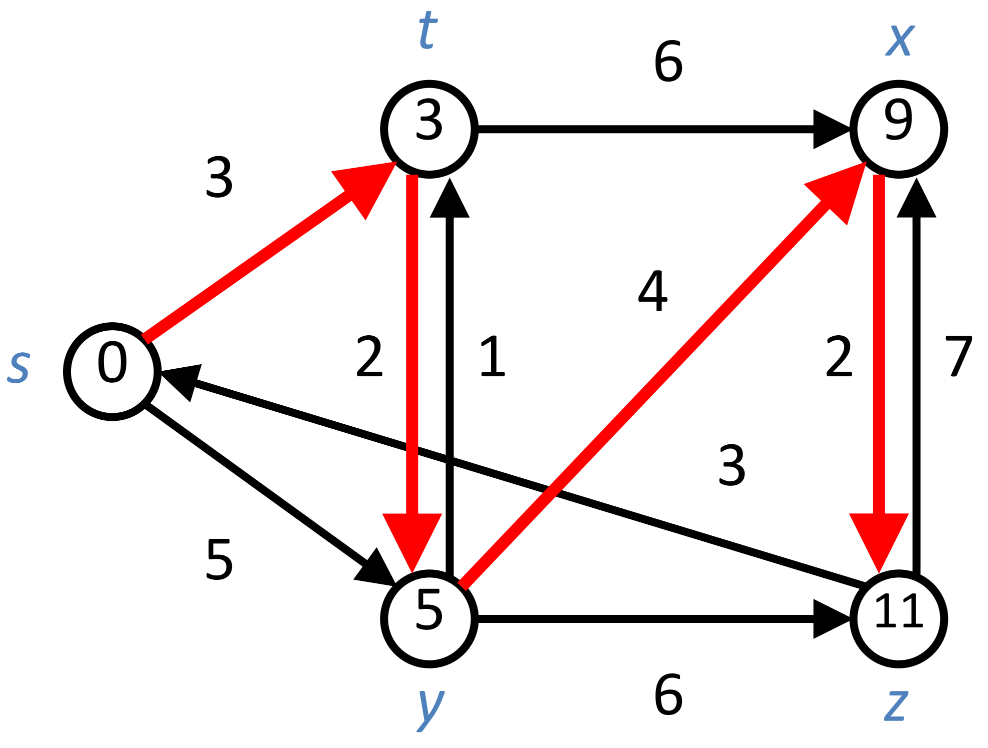
Shortest path from s?

Example



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

Example



Shortest paths are not necessarily unique!

Variants

- **Single-source:** 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.
- **Single-pair:** Find shortest path from u to v.

  Note: No way to known that is better in worst case than solving the single-source problem!
- **All-pairs:** Find shortest path from u to v for all $u, v \in V$.

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 $w(s, v) = -\infty$ 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.

Cycles

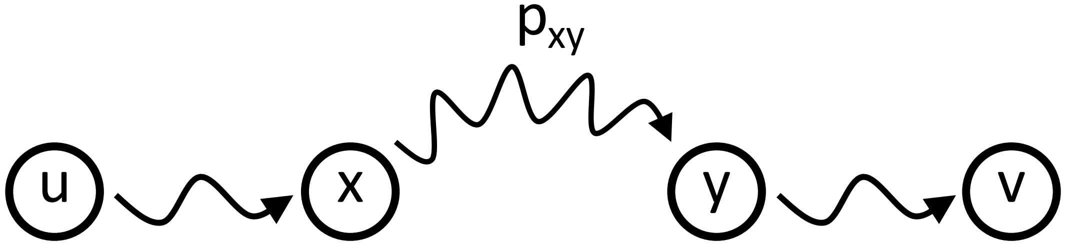
Shortest paths cannot contain cycles:

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

Optimal substructure

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

Proof (contradiction using cut and paste approach):



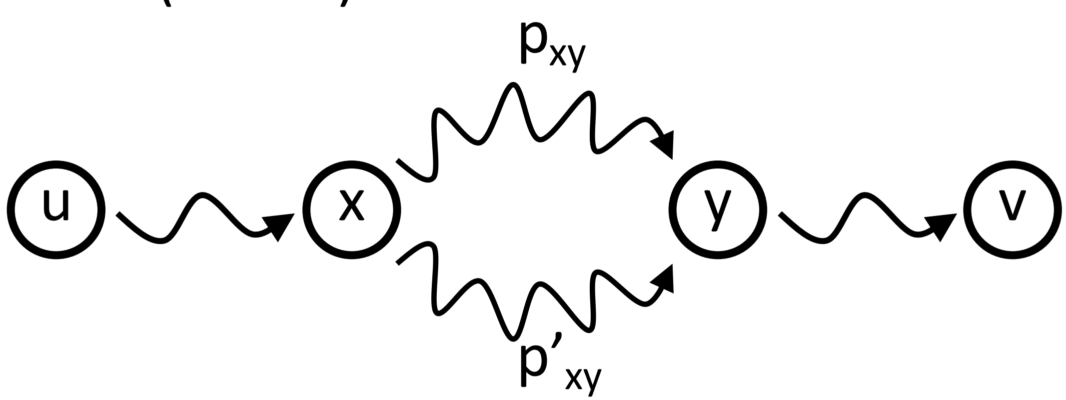
Suppose this path *p* is a shortest path from *u* to *v*.

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

Optimal substructure

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

Proof: (cont'd)



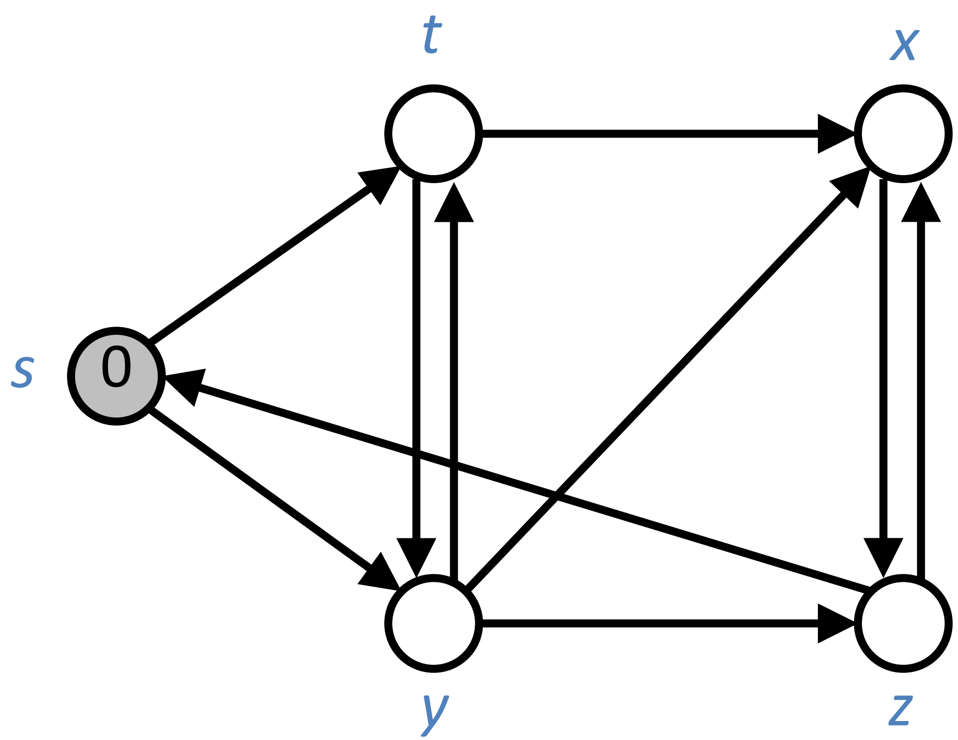
Now suppose there exists a shorter path $x \overset{p'_{xy}}{\rightsquigarrow} y$.

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

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

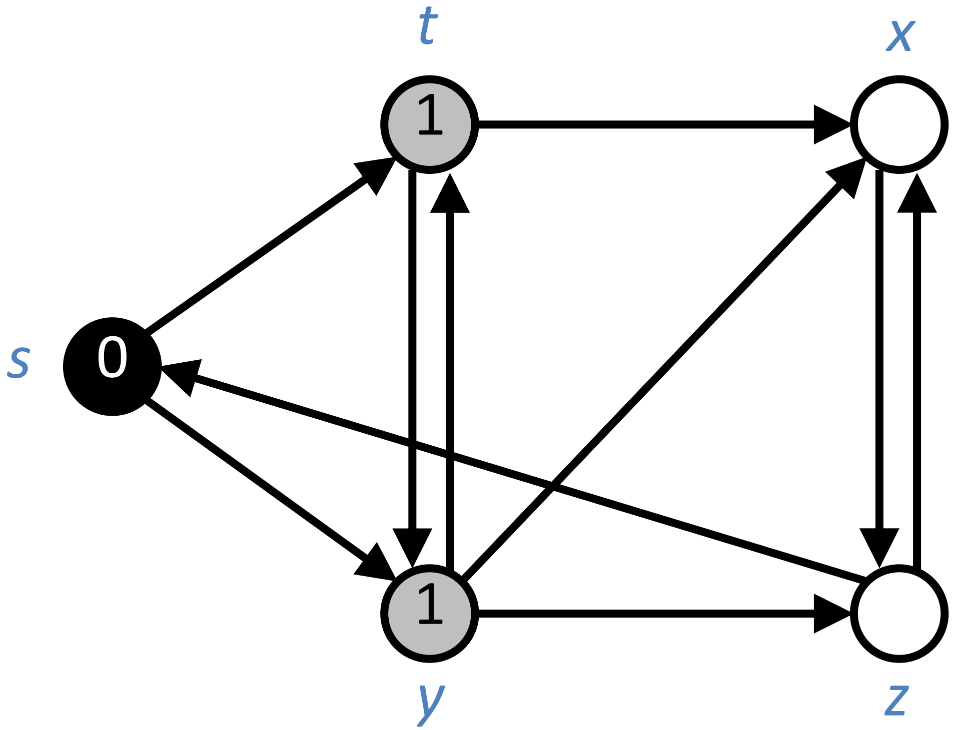
Contradiction of the hypothesis that p is the shortest path!

Customized breadth-first search

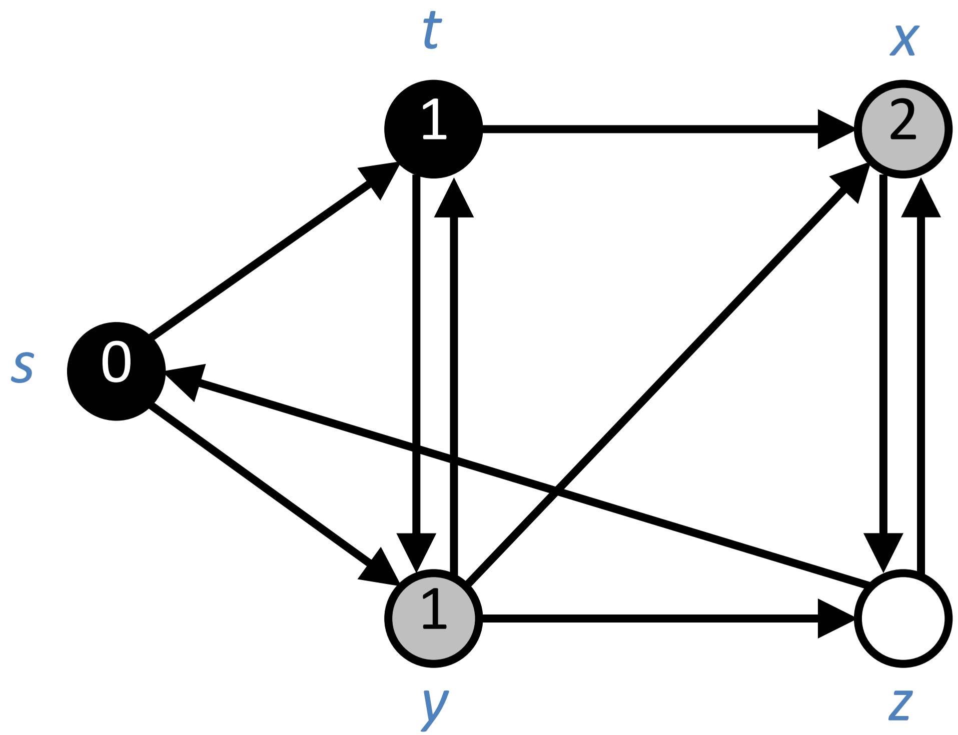


Vertices count the number of edges used to reach them.

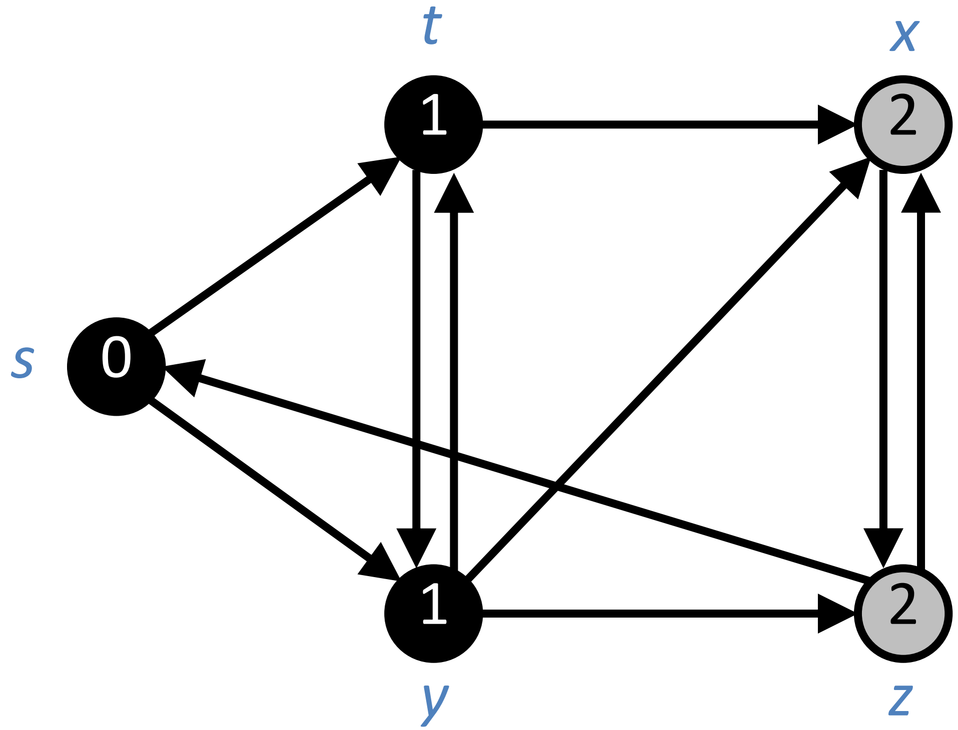
Customized breadth-first search



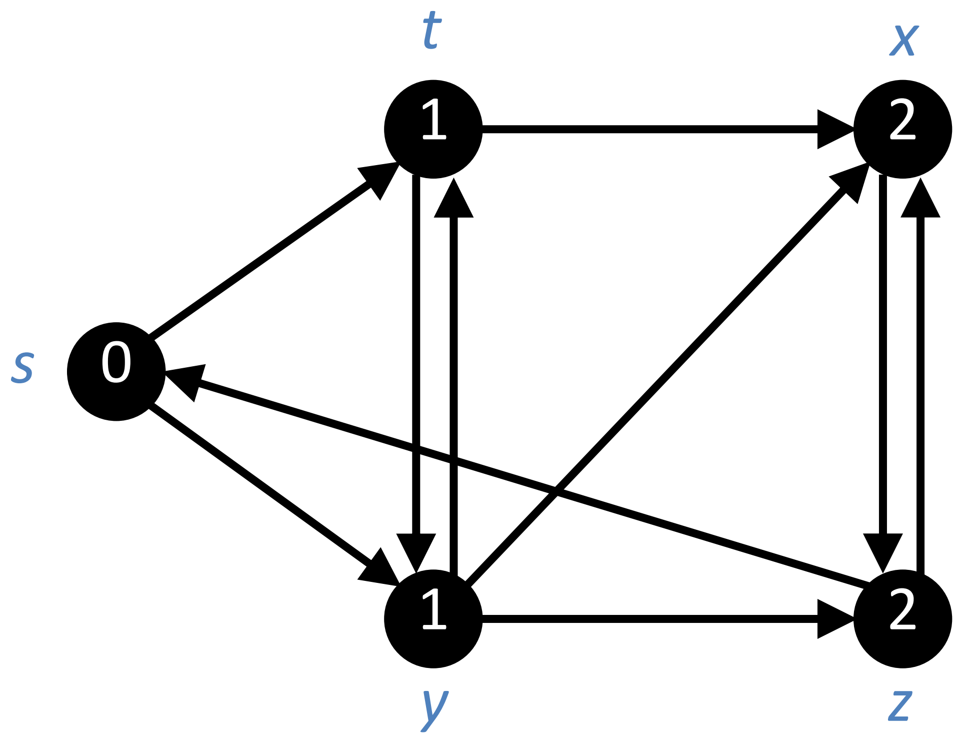
Customized breadth-first search



Customized breadth-first search



Customized breadth-first search



Can we generalize BFS to use edge weights?

Principle of a single-source shortest-path algorithm

For each vertex $v \in V$:

$\delta(s,v)$ is the absolute shortest path
d[v] is our current estimate of the shortest path

- $d[v] = \delta(s,v)$.
  - 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).

Generic algorithm structure

1. Initialization
2. Scan vertices and relax edges

The algorithms differ in the order and how many times they relax each edge.

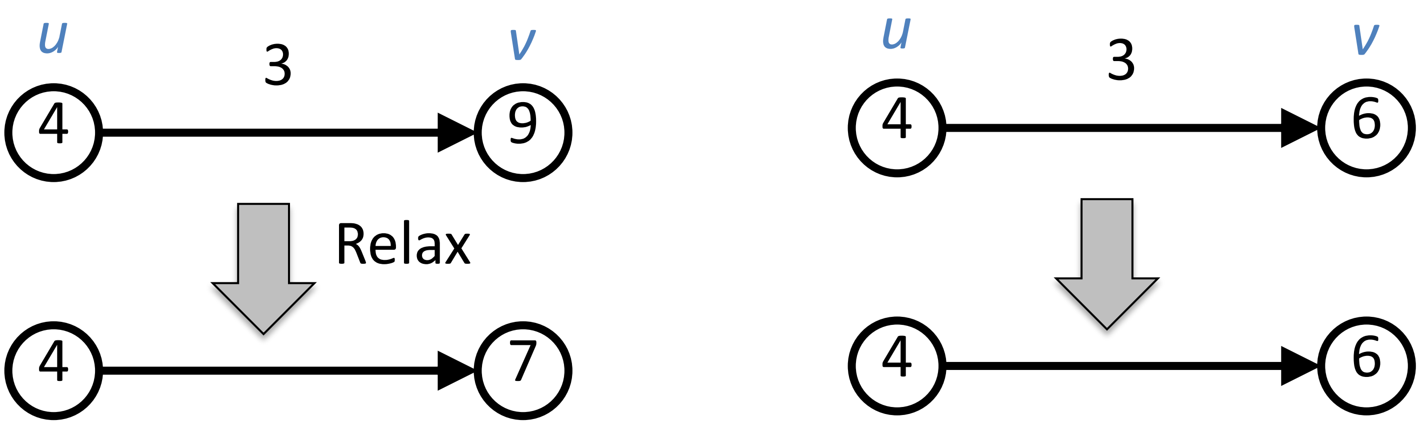
Initialization

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

Relaxing an edge

This is used to reduce d[v] during the execution of the algorithm.

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



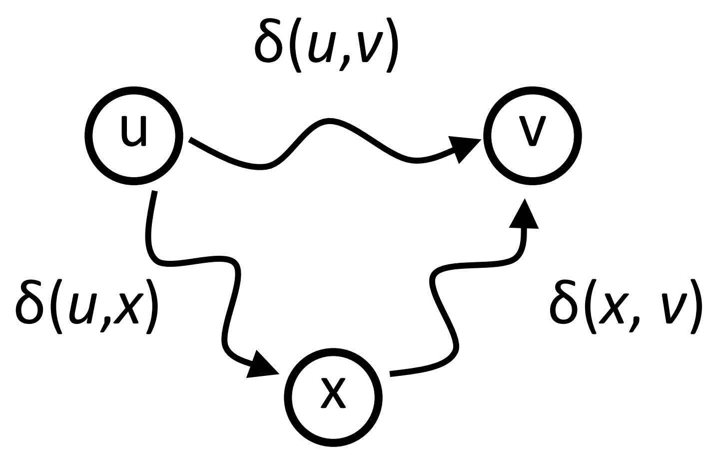
Triangle inequality

For all $(u,v) \in E$, we have $\delta(u,v) \leq \delta(u,x) + \delta(x,v)$.

Proof:

Weight of shortest path $u \rightsquigarrow$ v is ≤ weight of any path $u \rightsquigarrow$ v.

Path $u \rightsquigarrow x \rightsquigarrow$ v is a path $u \rightsquigarrow$ v, and if we use a shortest path $u \rightsquigarrow x$ *and* $x \rightsquigarrow v$, its weight is $\delta(u, x) + \delta(x, \mathrm{v})$.



Upper bound property

> Always have $\delta(s, \mathrm{v}) \leq d[\mathrm{v}]$ for all v.
> Once $d[\mathrm{v}] = \delta(s, \mathrm{v})$, it never changes.

Proof:

WLOG = Without Loss Of Generality

- Initially true.
- Then, assume it exists a vertex v such that $d[v] < \delta(s, v)$.
  WLOG, v is the first vertex for which this happens.
  Let u be the vertex that causes $d[v]$ to change.
  Then, after relaxation $d[v] = d[u] + \delta(u, v)$. But we also have:

$$d[v] < \delta(\mathrm{s,v}) \leq \delta(u, v) + \delta(s, u) \leq d[u] + \delta(u, v)$$

(triangle inequality)

(v is first violation, thus inequality is valid for u)

$\Rightarrow d[\mathrm{v}] < d[u] + \delta(u,v)$.

Note the strict inequality!

Contradicts $d[v] = d[u] + \delta(u, v)$.

No-path property

$$\boxed{\text{If } \delta(s, \mathrm{v}) = \infty, \text{ then } d[\mathrm{v}] = \infty \text{ always.}}$$

Proof: $d[\mathrm{v}] \geq \delta(s,\mathrm{v}) = \infty \Rightarrow d[\mathrm{v}] = \infty$.

Convergence property

We need all 3 conditions!

In 1: (u,v) is the last edge on the shortest path

If:
1. $s \rightsquigarrow u \to v$ is a shortest path,
2. $d[u] = \delta(s,u)$,
3. we call RELAX(u,v,w),

then $d[v] = \delta(s,v)$ afterward.

In other words, after RELAX d[v] is guaranteed to be the shortest path value.

Proof:

After relaxation:

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

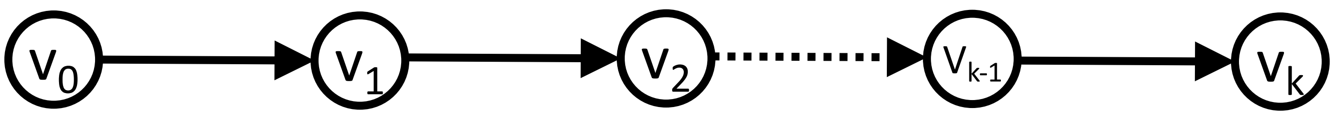
$= \delta(s, u) + w(u, v)$ ($d[u] = \delta(s,u)$)

$= \delta(s, v)$ ($s \rightsquigarrow u \to v$ is a shortest path & lemma sub-optimal structure)

Since $d[v] \geq \delta(s, v)$, must have $d[v] = \delta(s, v)$.

Path-relaxation property

Let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path from $s = v_0$ to v_k. If we relax, *in order*, (v_0,v_1), (v_1,v_2),..., (v_{k-1},v_k), even intermixed with other relaxations, then $d[v_k] = \delta(s,v_k)$.



Proof:

Induction to show that $d[v_i]=\delta(s,v_i)$ after (v_{i-1},v_i) is relaxed.

Basis: $i = 0$. Initially, $d[v_0] = 0 = \delta(s, v_0) = \delta(s,s)$.

Inductive step: Assume $d[v_{i-1}] = \delta(s,v_{i-1})$. Relax (v_{i-1},v_i). By *convergence property*, $d[v_i] = \delta(s, v_i)$ afterward and $d[v_i]$ never changes.

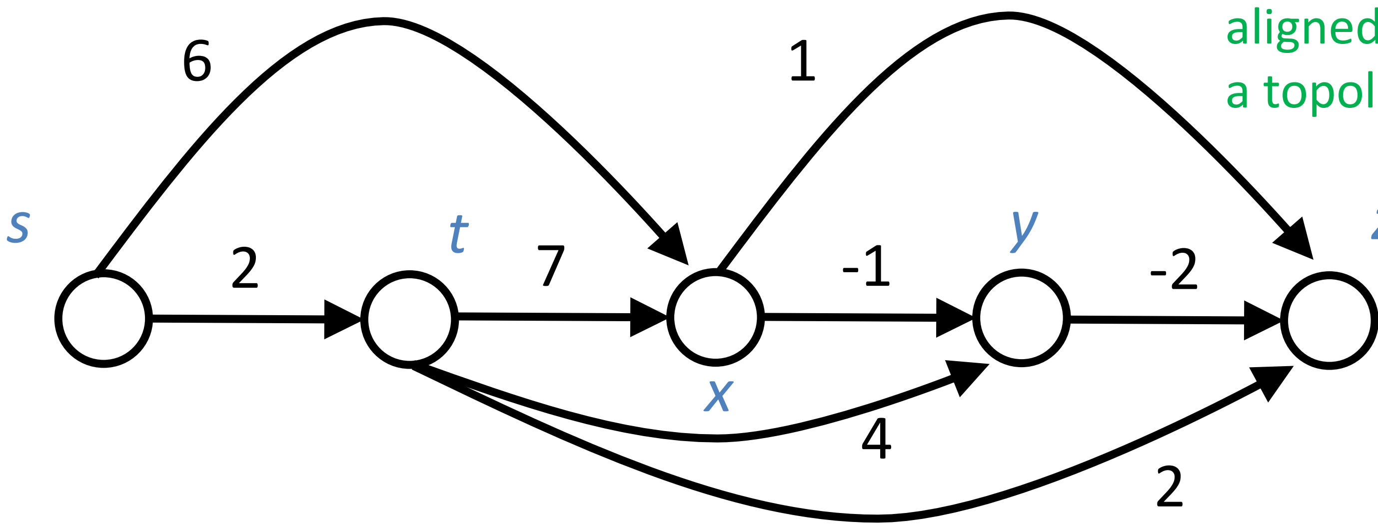
Single-source shortest paths in a DAG

DAG ⇒ no negative-weight cycles.

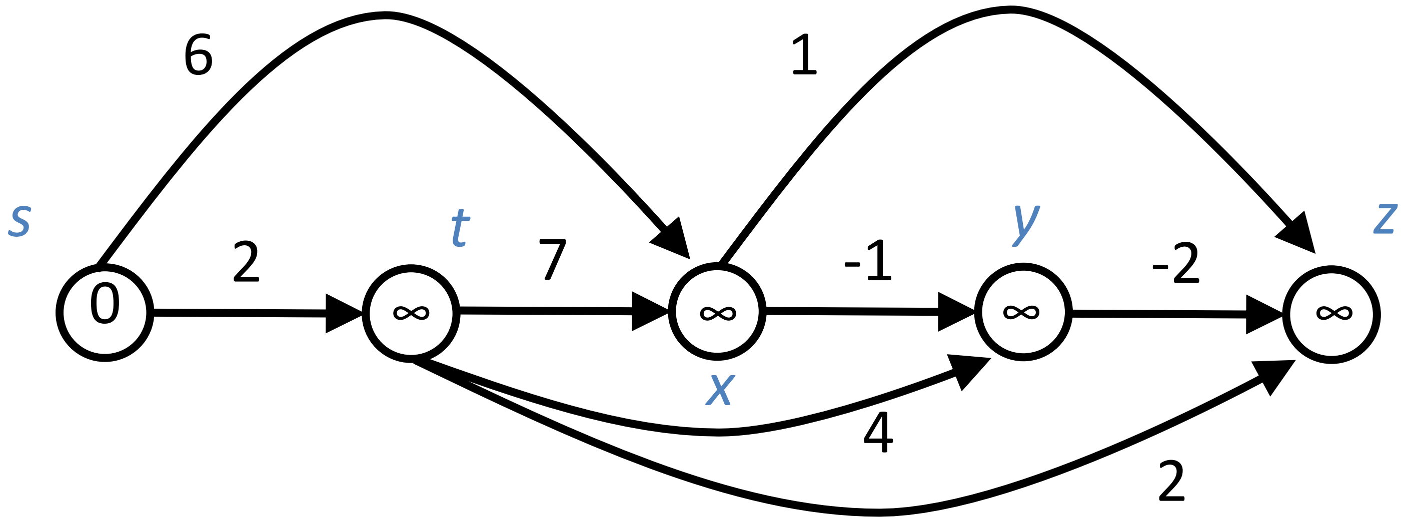
Note: the edges can have any weight you want.

```
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)
```

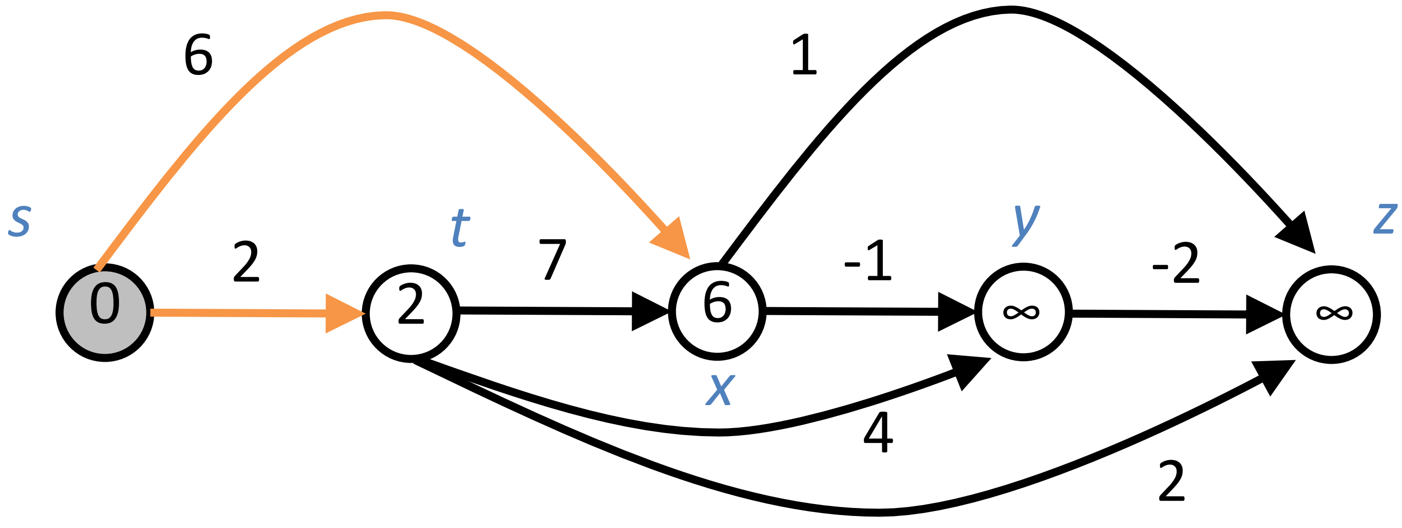
The vertices are aligned according to a topological order.



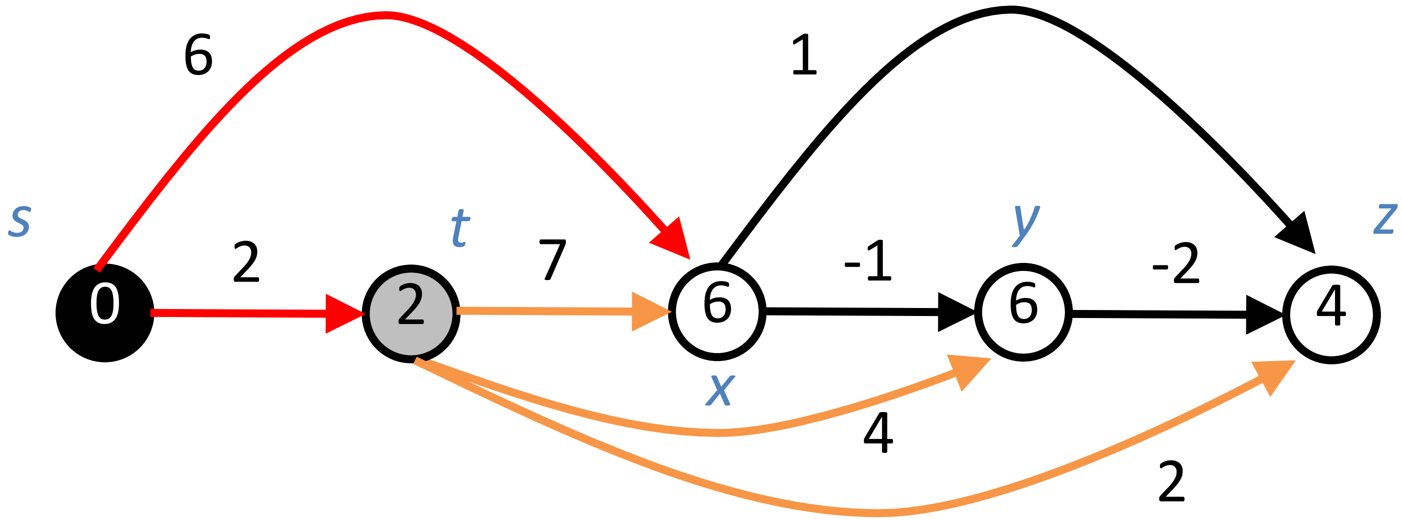
Example



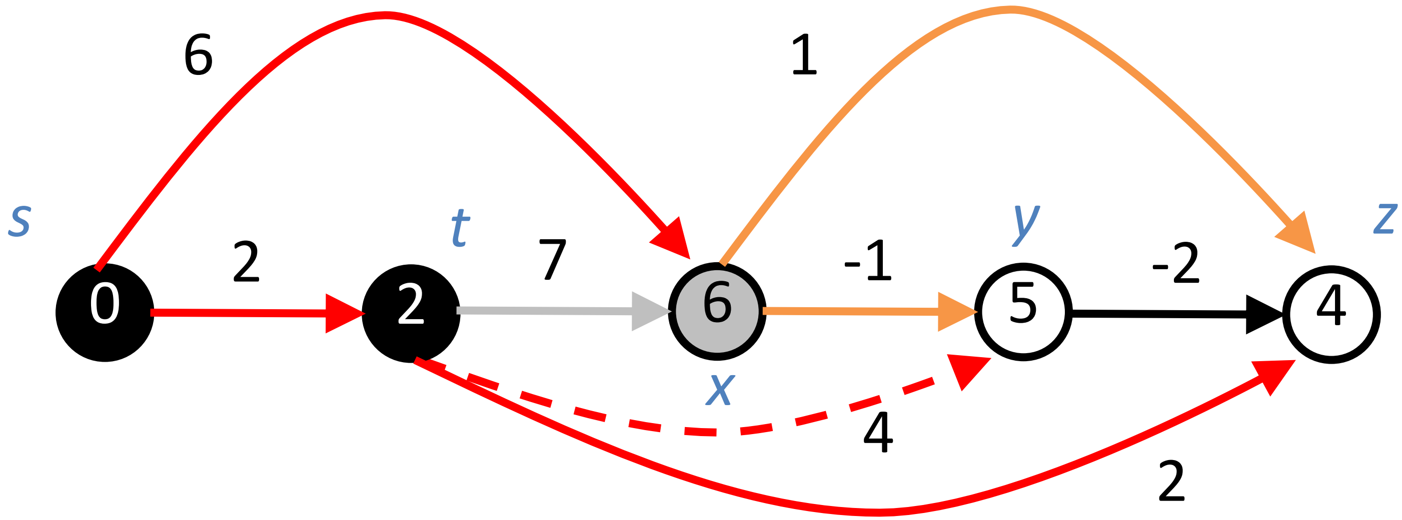
Example



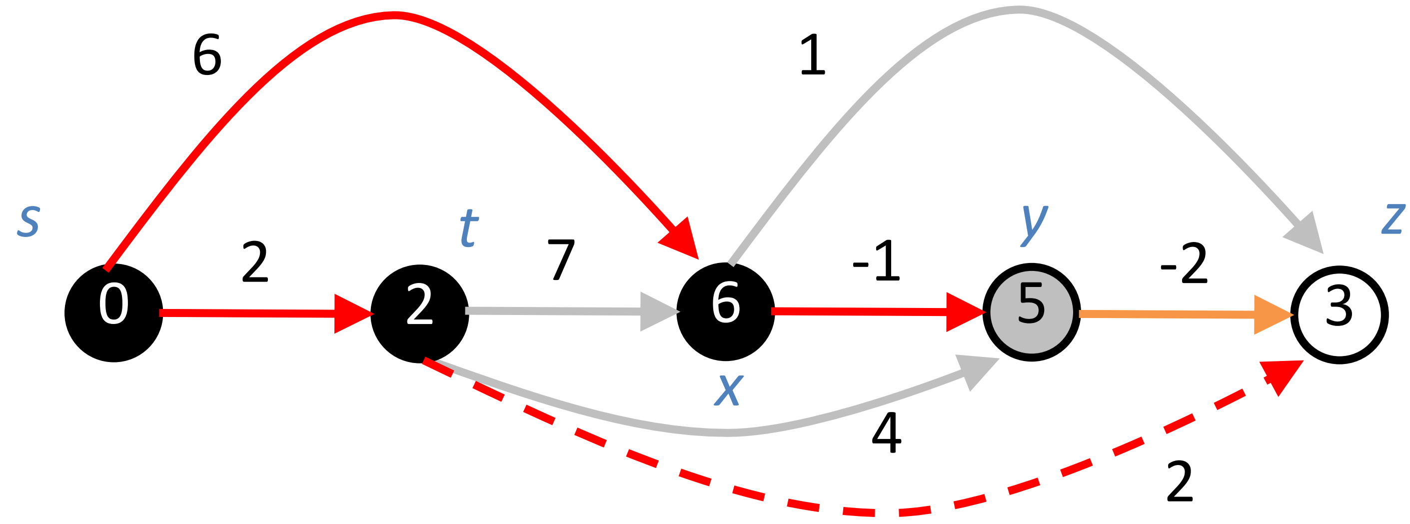
Example



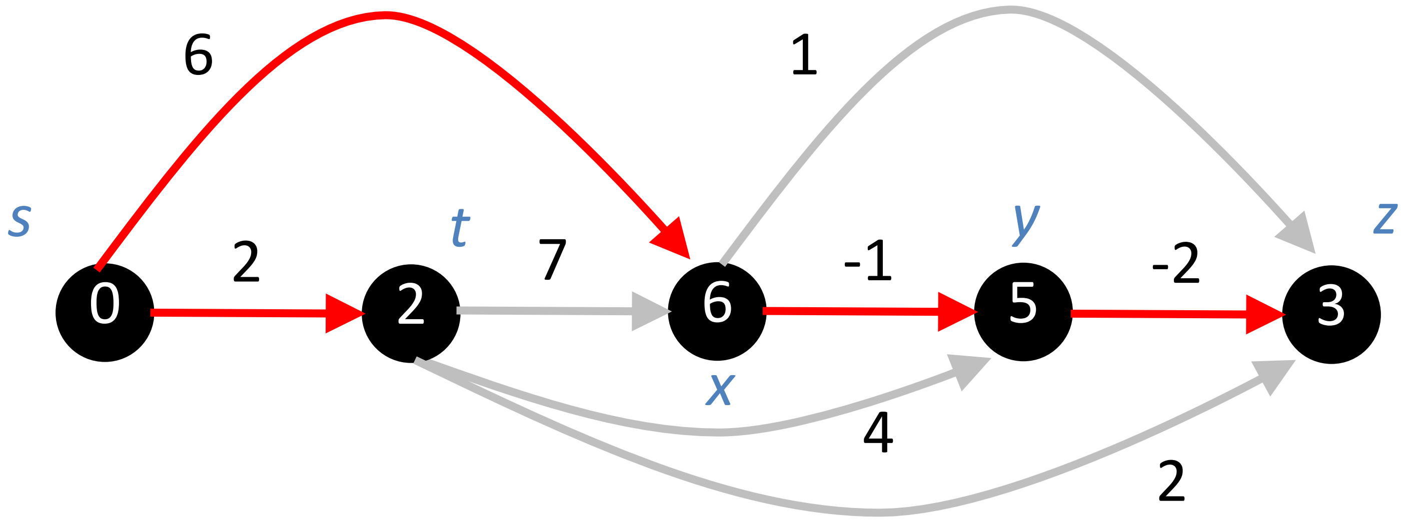
Example



Example



Example



Single-source shortest paths in a DAG

```
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)
```

Time: (V + E).

No edge leads you to a vertex already processed.

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 the path-relaxation property, the result is correct.

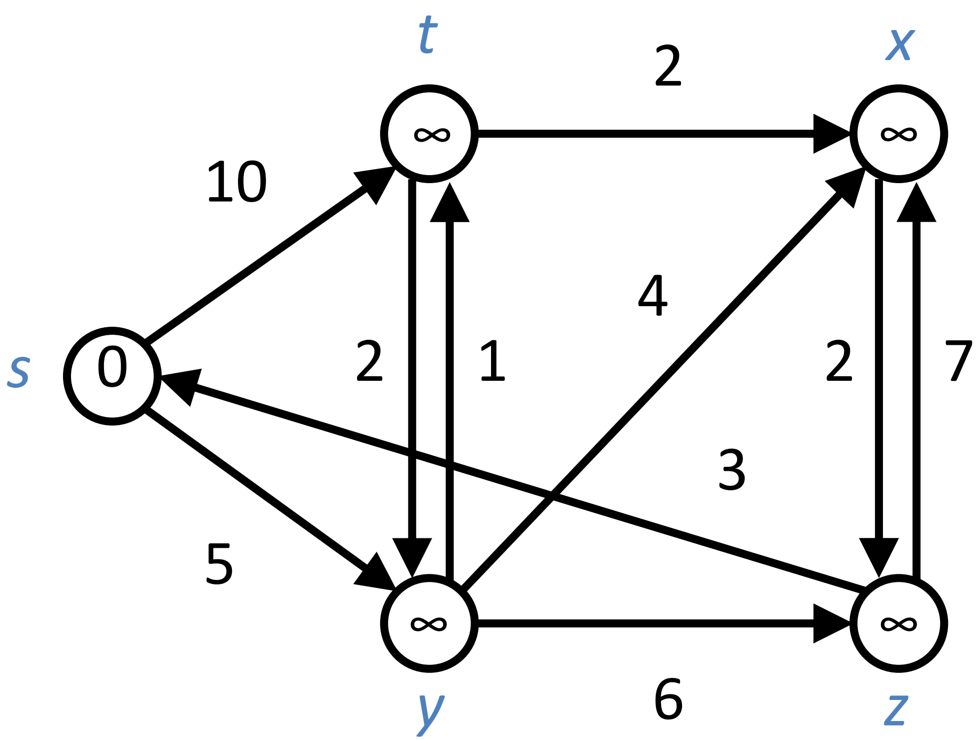
Dijkstra’s algorithm

- 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 = $V - S$.
- Similar Prim’s algorithm, but computing d[v], and using shortest-path weights as keys.
- Greedy choice: At each step we choose the light edge.

Dijkstra's algorithm

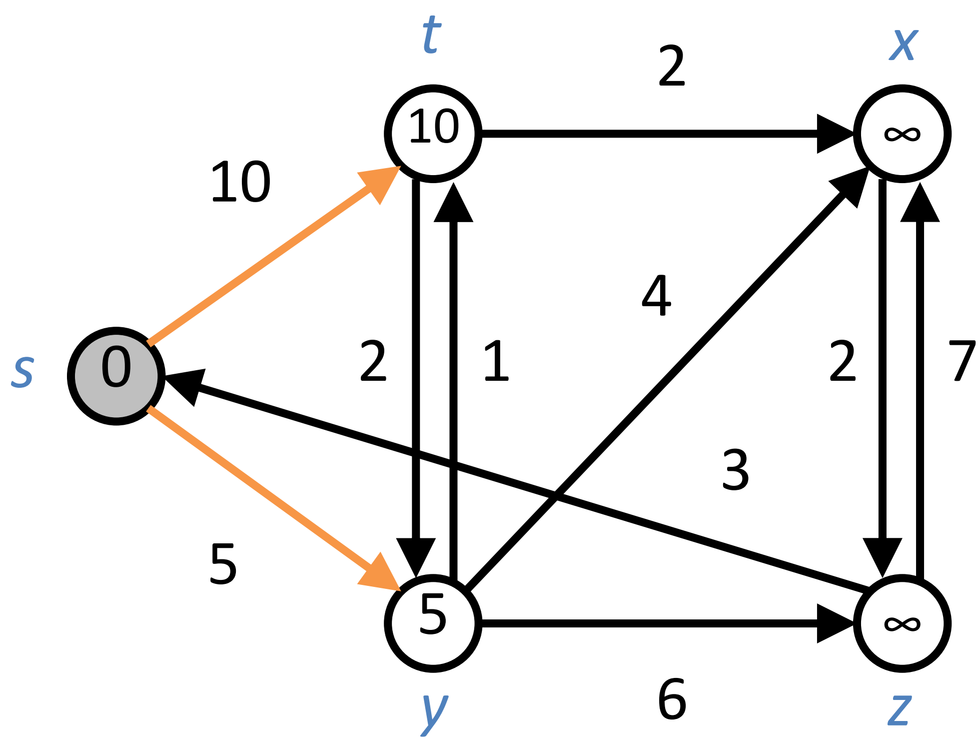
```
DIJKSTRA(V, E,w,s)
INIT-SINGLE-SOURCE(V,s)
S ← ∅
Q ← V
while Q ≠ ∅ do
   u ← EXTRACT-MIN(Q)
   S ← S ∪ {u}
   for each vertex v ∈ Adj[u] do
      RELAX(u,v,w)
```

Example



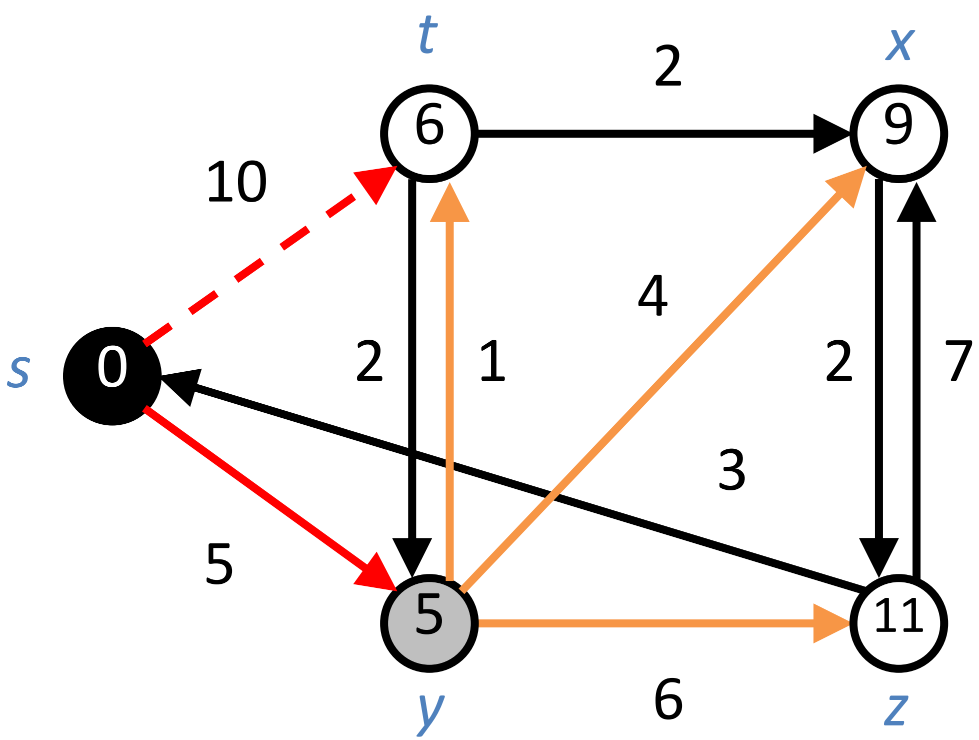
Q					
	s	t	y	x	z

Example



Q	y	t	x	z	

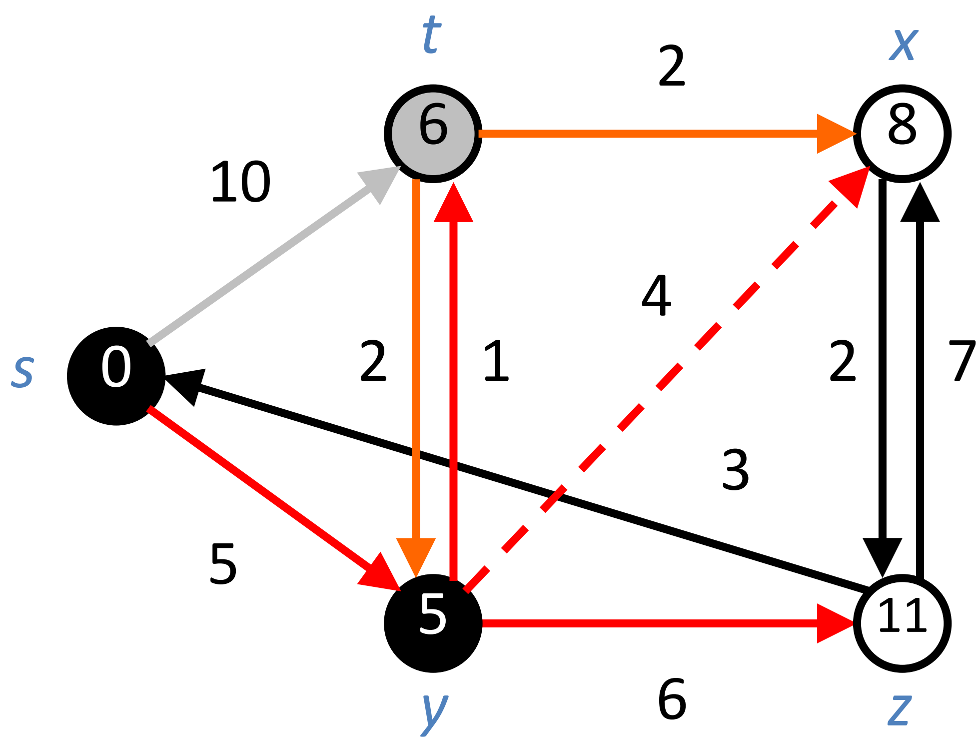
Example



Q

t	x	z		

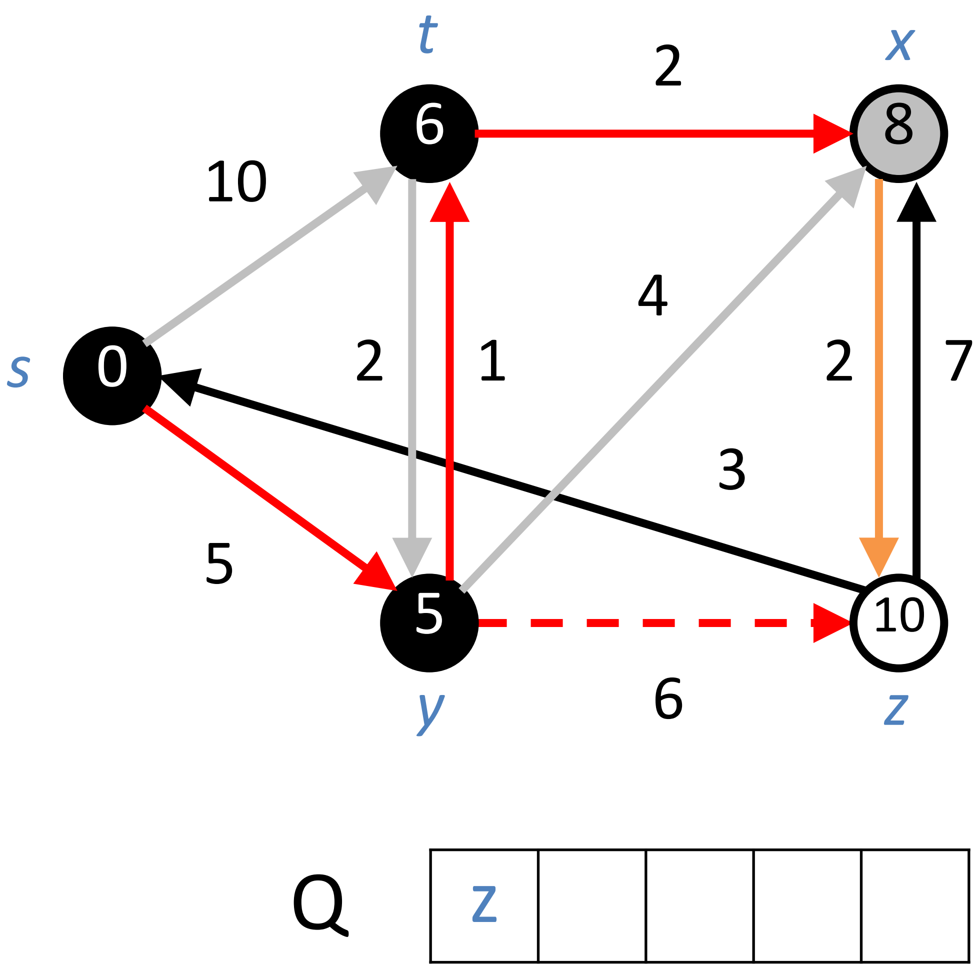
Example



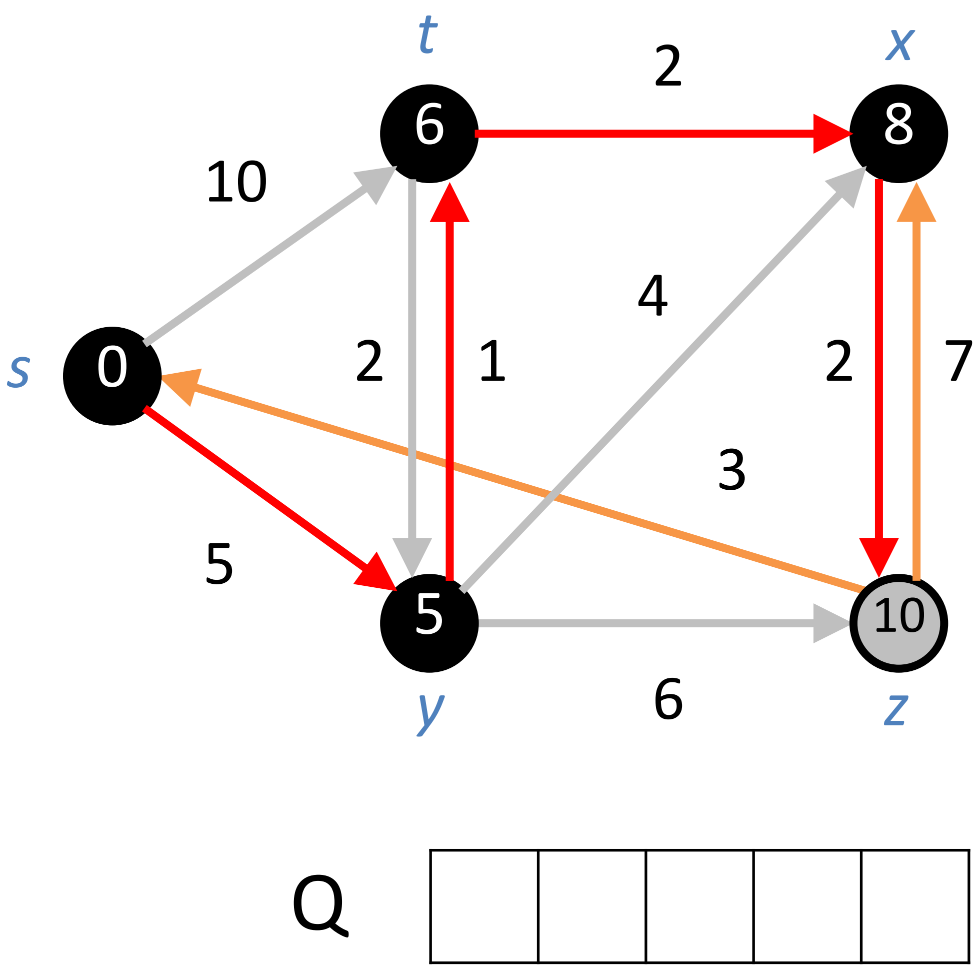
Q

x	z			

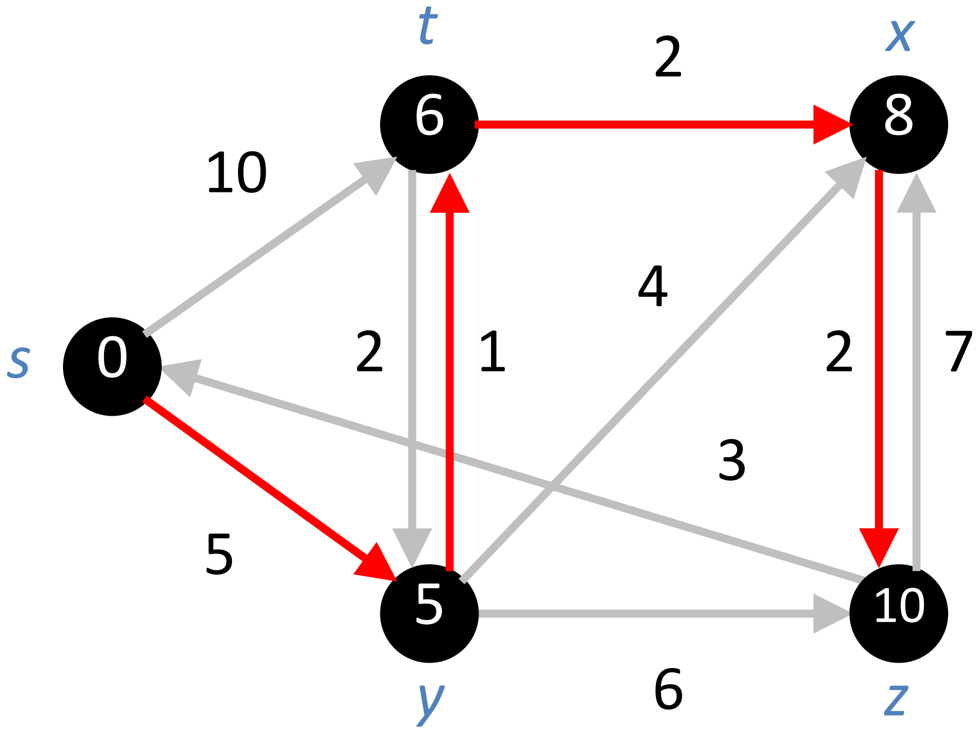
Example



Example



Example



Correctness

Loop invariant property:

At the end 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 each iteration, we remove one vertex from Q and never add one in. Thus, the algorithm terminates after $|V|$ iterations.

Maintenance:

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

Correctness (cont'd)

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, $s \in S$ *and* $S \neq \emptyset$.

Now, we can discuss the general case:

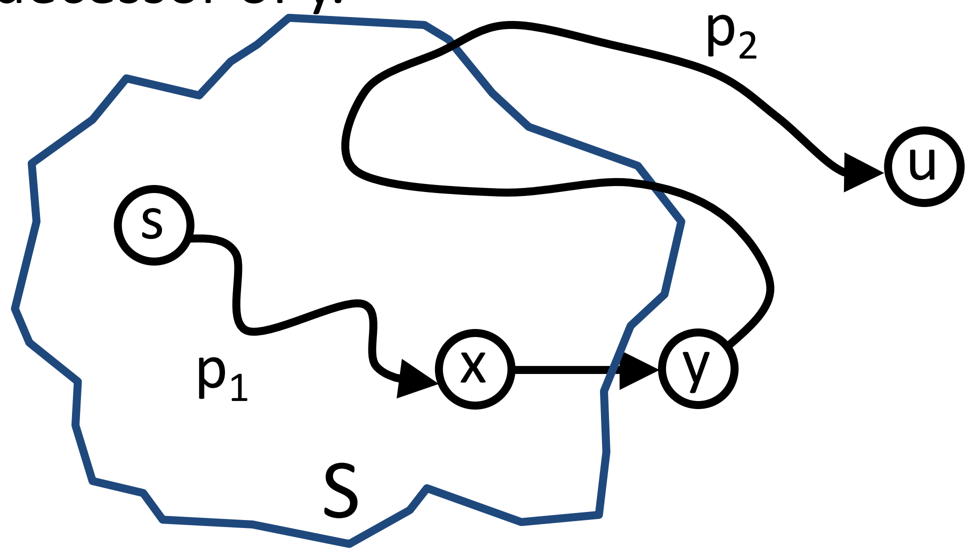
- There must be some path $s \rightsquigarrow u$. Otherwise $d[u] = \delta(s,u) = \infty$ by no-path property.
- Since there is a path $s \rightsquigarrow u$, there is also **a shortest *p* path $s \rightsquigarrow u$**.

Correctness (cont'd)

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

Just before u is added to S, the shorted 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.



Note: if y == u, then $d[u] = \delta(s,u)$ by the convergence property.

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

Correctness (cont'd)

We will need that claim to complete the proof

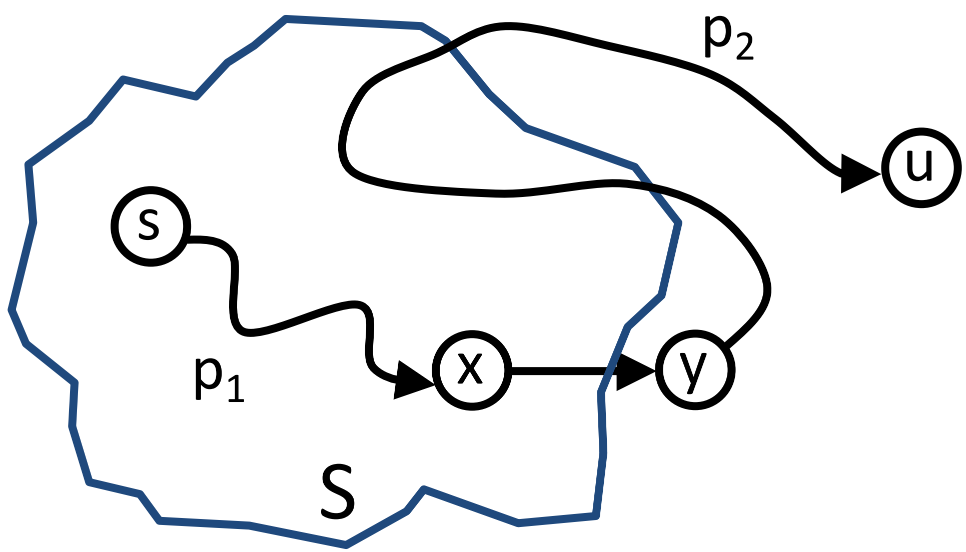
Claim 1: d[y] = δ(s, y) when u is added to S.

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).

We relax all outgoing edges of a vertex before adding it into S, and y is on the shortest path.



Correctness (cont'd)

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

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

$\leq \delta(s,u)$ (See inequality above)

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

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

We can now conclude:

This is why we use the priority queue!

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 $d[u] \neq \delta(s,u)$. ■

Correctness

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

Initialization: ✓

Termination: ✓

Maintenance: ✓

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

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.