Single-Source Shortest Paths

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)

Many applications

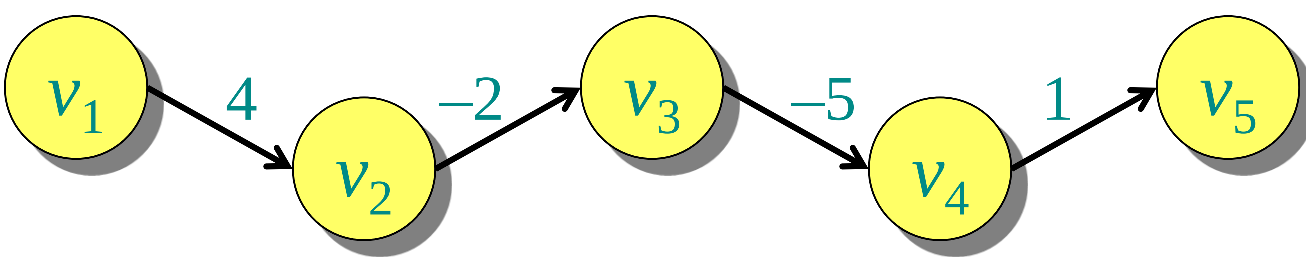
- Shortest paths model many useful real-world problems.
  - Minimization of latency in the Internet.
  - Minimization of cost in power delivery.
  - Job and resource scheduling.
  - Route planning.

Paths in graphs

- Consider a directed graph $G = (V, E)$ with edge-weight function $w : E \to \mathbb{R}$. The ***weight*** of path $p = v_1 \to v_2 \to \dots \to v_k$ is defined to be

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

Example:



$w(p) = -2$

Shortest paths

- A ***shortest path*** from u to v is a path of minimum weight from u to v. The ***shortest-path weight*** from u to v is defined as

$$\mathrm{d}(u, v) = \begin{cases} \min\{w(p) : p \text{ is a path from } u \text{ to } v\} \\ \infty \text{ if no path from } u \text{ to } v \text{ exists} \end{cases}$$

Optimal substructure

Lemma 24.1: Subpaths of shortest paths are shortest paths

Given a weighted, directed graph $G = (V, E)$ with weight function $w: E \rightarrow R$, let $p = (v_0, v_1, \ldots\ldots, v_k)$ be a shortest path from vertex v_0 to vertex v_k and for any *i* and *j* such that $0 \le i \le j \le k$, let $p_{ij} = (v_i, v_{i-1}, \ldots\ldots, v_j)$ *i* be the subpath of p from vertex *i* to vertex *j* . Then, p_{ij} is a shortest path from *i* to *j* .

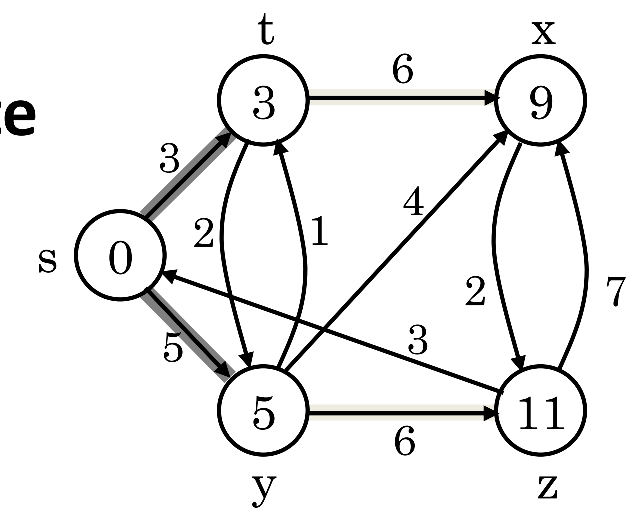
Proof: If we decompose path p into $v_0 \xrightarrow{p_{0i}} v_i \xrightarrow{p_{ij}} v_j \xrightarrow{p_{jk}} v_k$, then we have that $w(p) = w(p_{0i}) + w(p_{ij}) + w(p_{jk})$.

Assume that there is a path p'_{ij} from i to j with weight $w(p'_{ij}) < w(p_{ij})$. Then, $v_0 \xrightarrow{p_{0i}} v_i \xrightarrow{p'_{ij}} v_j \xrightarrow{p_{jk}} v_k$ is a path from 0 to k whose weight $w(p_{0i}) + w(p'_{ij}) + w(p_{jk})$ is less than w(p), which contradicts the assumption that p is a shortest path from 0 to k.

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

INITIALIZE-SINGLE-SOURCE(V, s)

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

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

3. π[v] $\leftarrow$ NIL

4. d[s] $\leftarrow$ 0

All the shortest-paths algorithms start with INITIALIZE-SINGLE-SOURCE

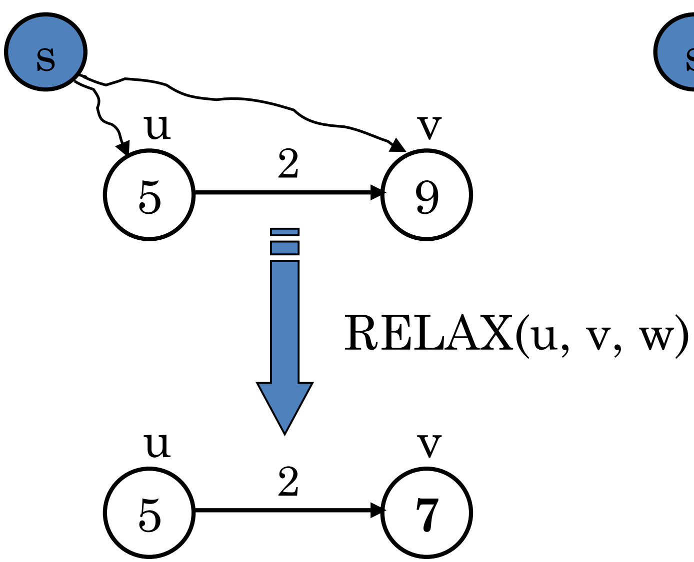
Relaxation

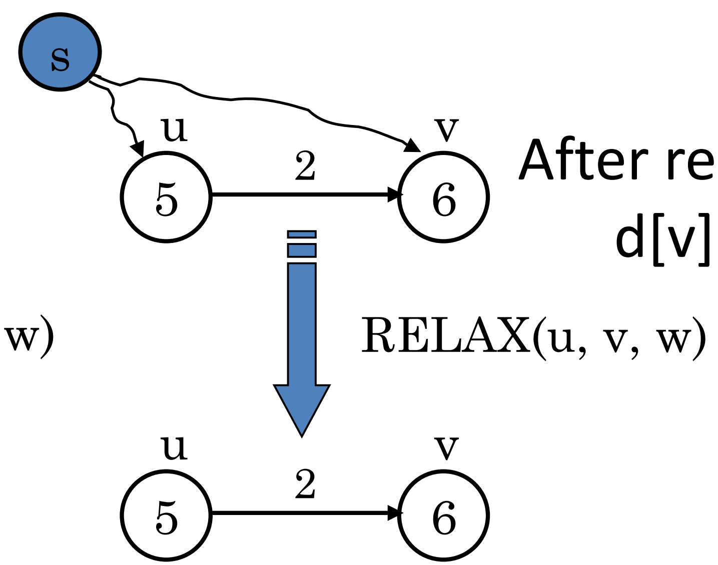
- **Relaxing** an edge (u, v) = testing whether we can improve the shortest path to v found so far by going through u

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

  we can improve the shortest path to v

  $\Rightarrow$ update d[v] and π[v]





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

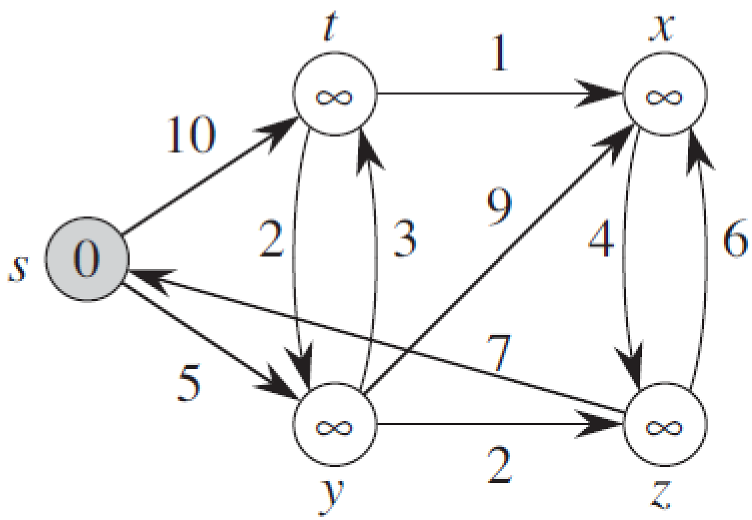
Dijkstra's Algorithm

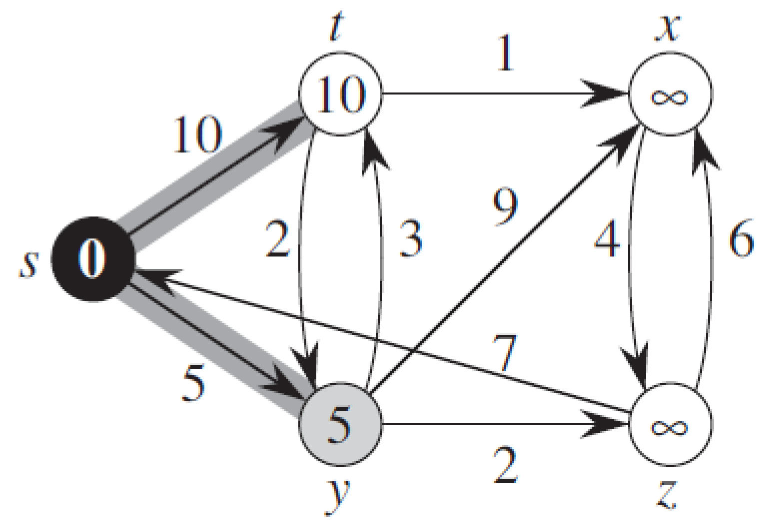
- Single-source shortest path problem:
  - No negative-weight edges: $w(u, v) > 0 \ \forall \ (u, v) \in E$
- Maintains two sets of vertices:
  - S = vertices whose final shortest-path weights have already been determined
  - Q = vertices in V – S: min-priority queue
    - Keys in Q are estimates of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

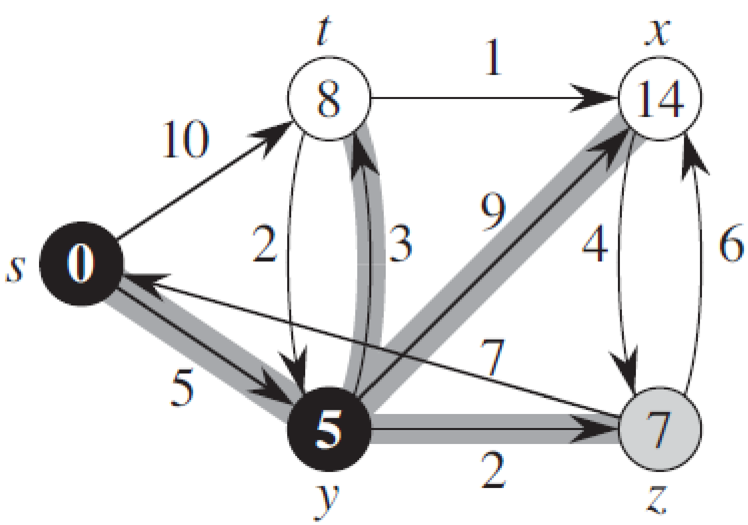
Dijkstra (G, w, s)

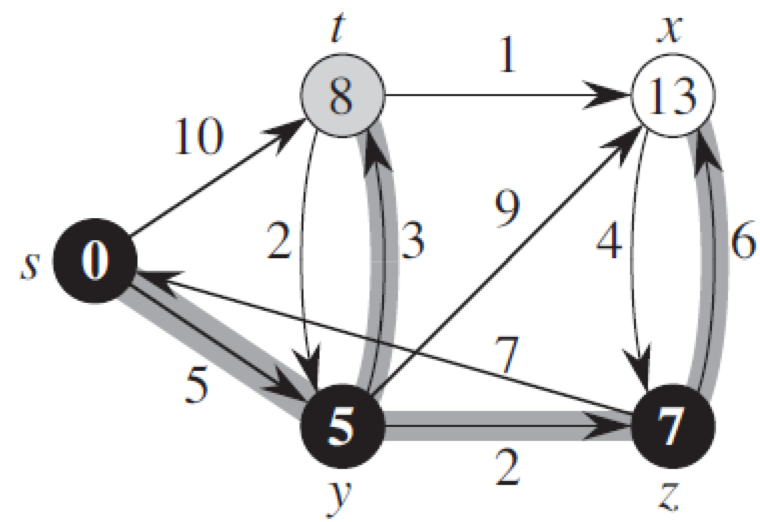
```
INITIALIZE-SINGLE-SOURCE(V, s)
S ← ∅
Q ← V[G]
while Q ≠ ∅
    u ← EXTRACT-MIN(Q)
    S ← S ∪ {u}
    for each vertex v ∈ Adj[u]
        RELAX(u, v, w)
```

Dijkstra’s Algorithm(Example)

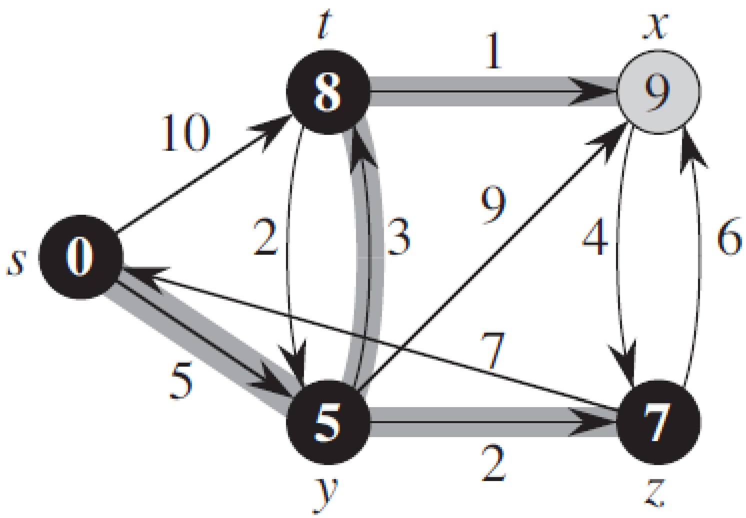


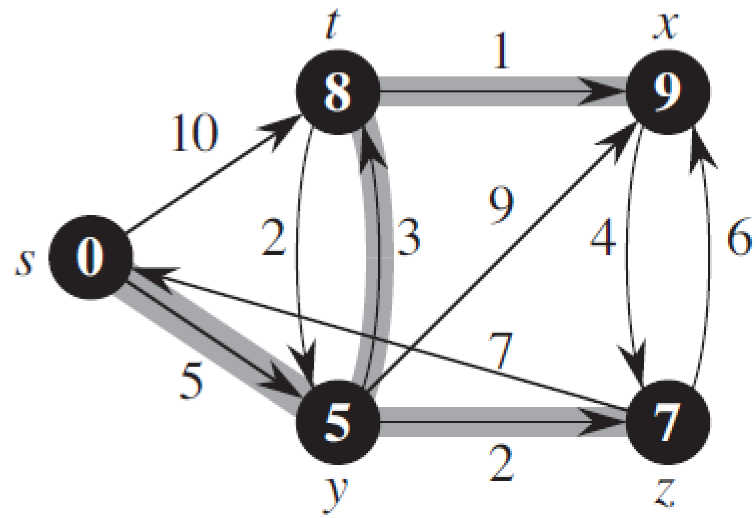






Dijkstra’s Algorithm(Example)





Dijkstra (G, w, s)

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

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

Analysis of Dijkstra's Algorithm

$$\text{Time} = \Theta(n)\cdot T_{\text{ExtractMin}} + \Theta(m)\cdot T_{\text{ChangeKey}}$$

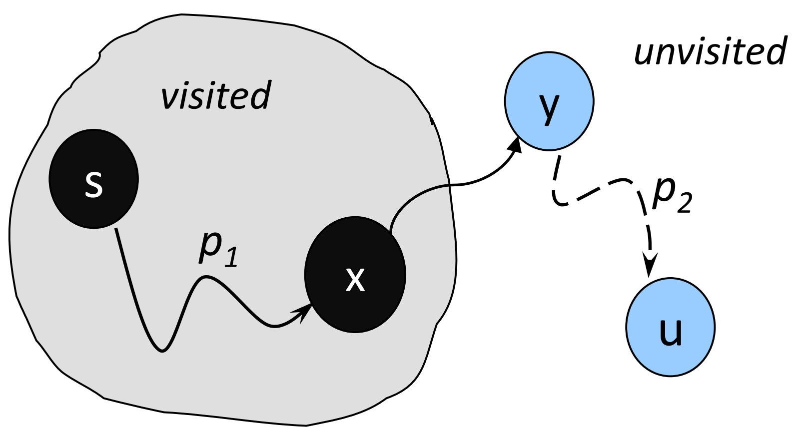
Q	$T_{\text{ExtractMin}}$	$T_{\text{ChangeKey}}$	Total
array	$\Theta(n)$	$\Theta(1)$	$\Theta(n^2)$
Priority queue	$\Theta(\log n)$	$\Theta(\log n)$	$\boxed{\Theta(m \log n)}$

Correctness (Dijkstra's Algorithm)

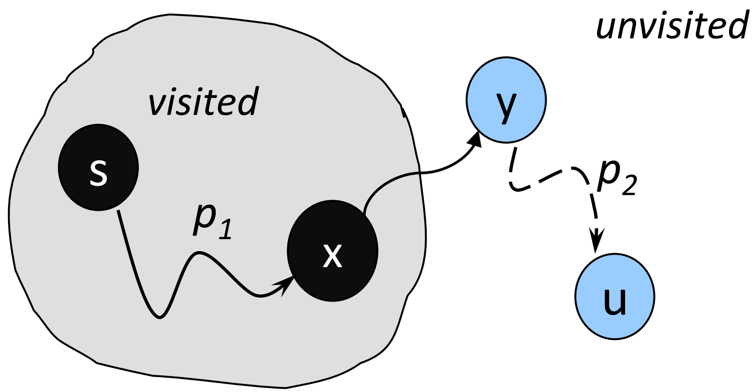
Prove in each iteration, u.d = δ(s, u) for the vertex added to set S(visited).

Let u be the first vertex for which u.d ≠ δ(s, u) when it is added to set S.

Let us consider the first vertex y along **p**(s $\overset{p_1}{\rightarrow}$ x $\rightarrow$y $\overset{p_2}{\rightarrow}$ u) such that y ϵ V - S, and let x ϵ S be y's predecessor along **p**.



Correctness (Dijkstra's Algorithm)



We claim that $y.d = \delta(s, y)$ when u is added to S. Because we had $x.d = \delta(s, x)$ when x is added to S. Edge(x, y) was relaxed at that time.

As y appears before u on a shortest path from s to u, we have $\delta(s, y) \le \delta(s, u)$

$y.d = \delta(s, y)$

$\le \delta(s, u)$

$\le u.d$

But because both vertices u and y were in V-S when u was chosen by extracting min from the heap we have $u.d \le y.d$

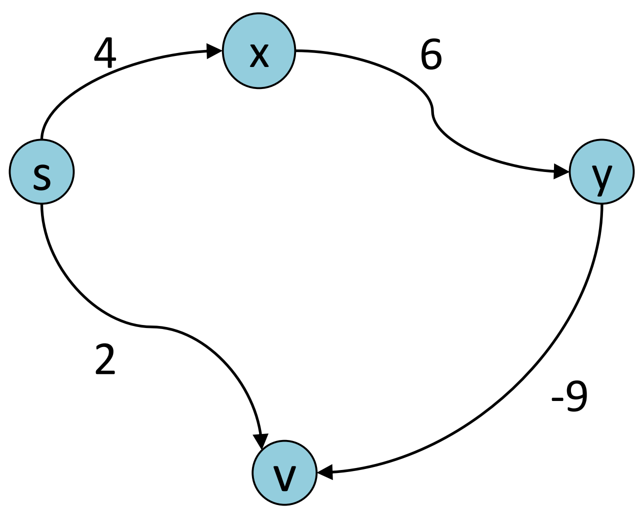
Therefore, $y.d = \delta(s, y) = \delta(s, u) = u.d$ which contradicts our choice of u.

Dijkstra’s Algorithm - negative weights?

Dijkstra’s Algorithm fails if there are negative weights.

Example: Select vertex v immediately after s

But shortest path from s to v is s-x-y-v



Bellman-Ford Idea

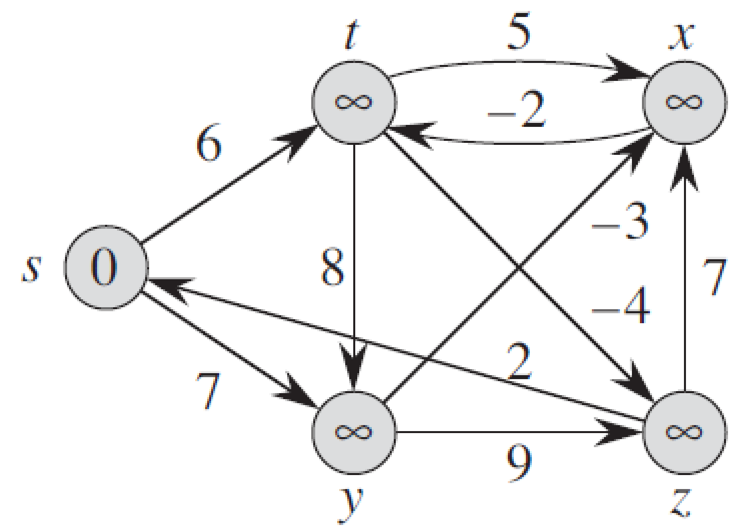
- Consider each edge (u,v) and see if u offers v a cheaper path from s
    - compare d[v] to d[u] + w(u,v)
- Repeat this process |V| - 1 times to ensure that accurate information propgates from s, no matter what order the edges are considered in

Bellman-Ford Algorithm

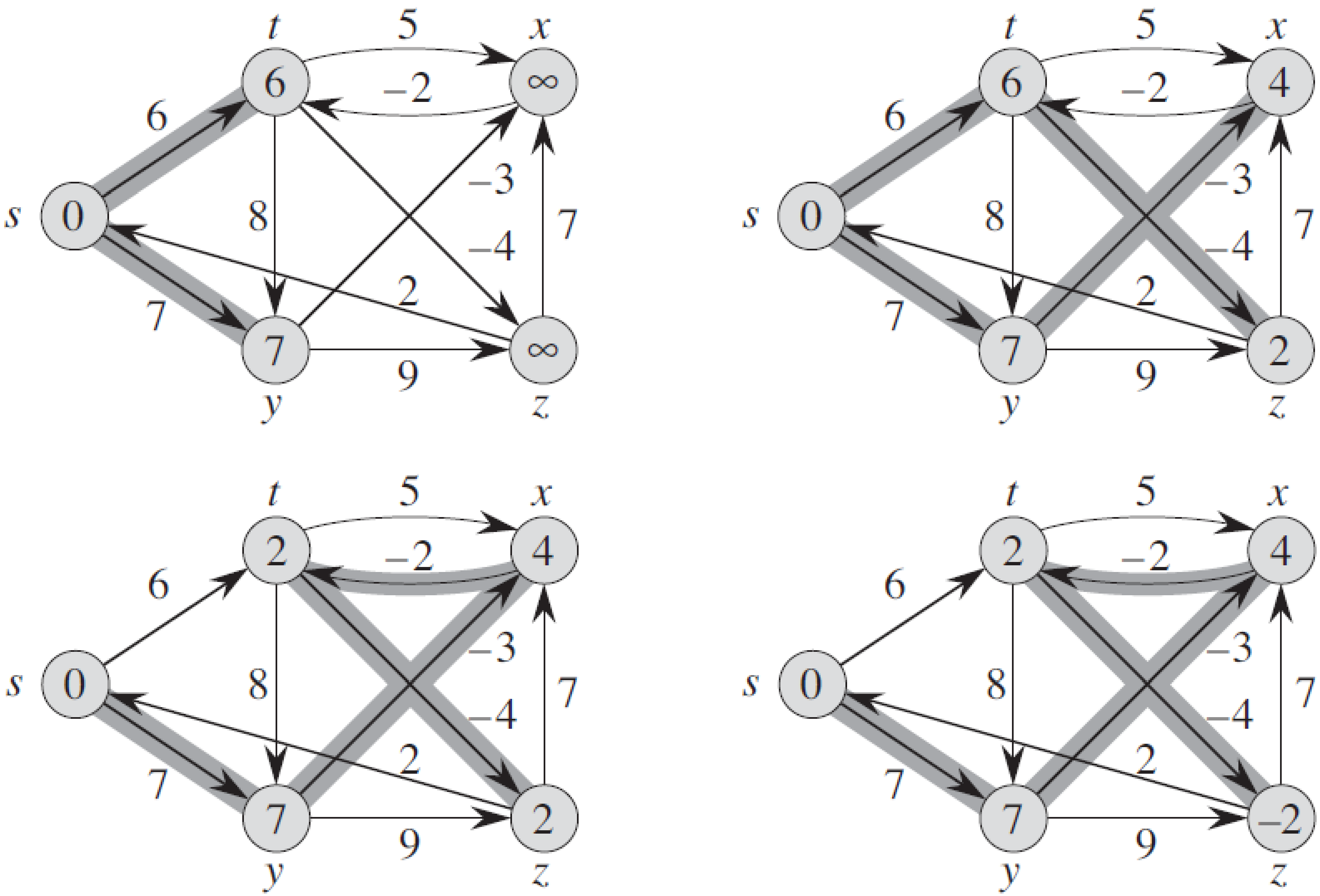
- Single-source shortest paths problem
  - Computes d[v] and π[v] for all $v \in V$
- Allows negative edge weights
- Returns:
  - TRUE if no negative-weight cycles are reachable from the source s
  - FALSE otherwise $\Rightarrow$ no solution exists
- Idea:
  - Traverse all the edges $|V – 1|$ times, every time performing a relaxation step of each edge

Bellman-Ford(G, w, s)

INITIALIZE-SINGLE-SOURCE(G, s)
for i ← 1 to |G.V| - 1
    do for each edge (u, v) ∈ G.E
        do RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    do if d[v] > d[u] + w(u, v)
        then return FALSE
return TRUE

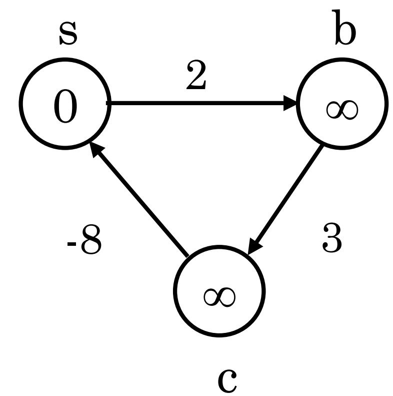


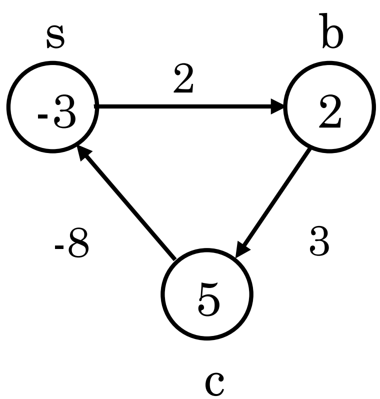
Bellman-Ford(Example)

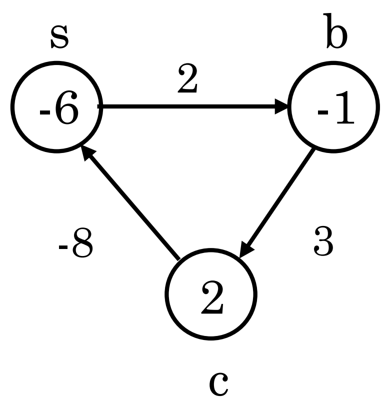


Detecting Negative Cycles

for each edge (u, v) $\in$ E
  do if d[v] > d[u] + w(u, v)
    then return FALSE
return TRUE







Look at edge (s, b):

d[b] = -1
d[s] + w(s, b) = -4

$\Rightarrow$ d[b] > d[s] + w(s, b)

Bellman-Ford(G, w, s)

```
INITIALIZE-SINGLE-SOURCE(G, s)              ← Θ(V)
for i ← 1 to |G.V| - 1                      ← O(V)   ⎫
    do for each edge (u, v) ∈ G.E           ← O(E)   ⎬ O(VE)
        do RELAX(u, v, w)                            ⎭
for each edge (u, v) ∈ G.E                  ← O(E)
    do if d[v] > d[u] + w(u, v)
        then return FALSE
return TRUE
```

Running time: O(VE)

Correctness (Bellman-Ford)

If G does contain a negative-weight cycle reachable from s, then the algorithm returns FALSE.

Graph ***G=*(*V,E*)** contains a negative-weight cycle c = **< *v_0, v_1,..., v_k*>** reachable from the source vertex ***s*** where $v_0 = v_k$. Then,

$$\sum_{i=1}^{k} w(v_{i\,-\,1}, v_i) < 0$$

Assume for the purpose of contradiction that the Bellman-Ford algorithm returns TRUE.

Thus, $v_i.d \le v_{i-1}.d + w(v_{i-1}, v_i)$ for i = 1,2,…,k

Correctness (Bellman-Ford)

$$\sum_{i=1}^{k} v_i.d \leq \sum_{i=1}^{k} (v_{i\ -\ 1}.d + \ w(v_{i\ -\ 1}, v_i))$$

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

Since $v_0 = v_k$, each vertex in c appears exactly once in each of the summations.

$$\sum_{i=1}^{k} v_i.d = \sum_{i=1}^{k} v_{i\ -\ 1}.d$$

$$\sum_{i=1}^{k} w(v_{i\ -\ 1}, v_i) > 0$$

which contradicts previous inequality.