Algorithms
Chapter 24
Single-Source Shortest Paths

Associate Professor: Ching-Chi Lin
林清池 副教授
chingchi.lin@gmail.com

Department of Computer Science and Engineering
National Taiwan Ocean University

Outline

- **The Bellman-Ford algorithm**
- Single-source shortest paths in directed acyclic graphs
- Dijkstra's algorithm

Single-source shortest paths problem

- **Input:** A weighted graph $G = (V, E)$ and a **source** vertex s.
- **Output:** Find a shortest path from s to every vertex $v \in V$.
- The weight $w(p)$ of path $p = \langle v_0, v_1, \ldots, v_k \rangle$ is the sum of the weights of its constituent edges:

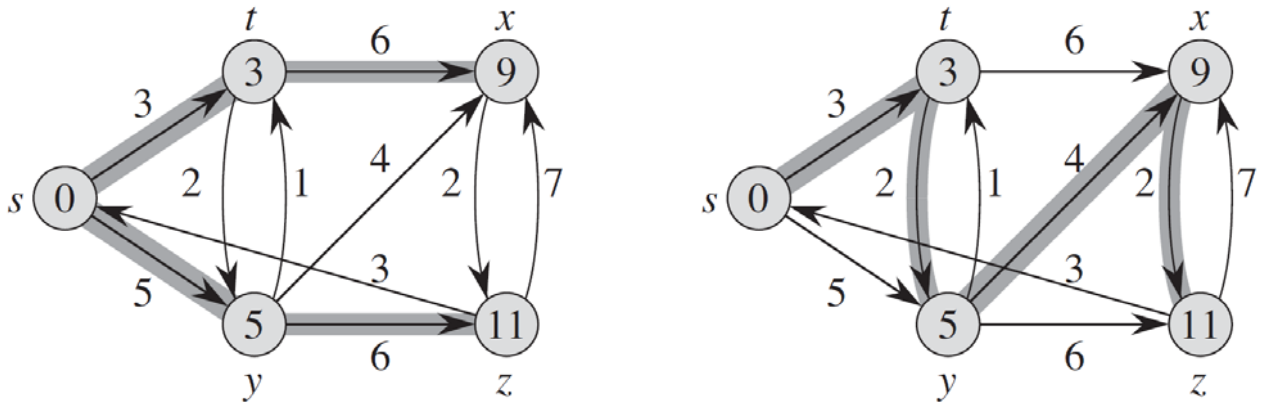
$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i).$$

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

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

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

An example

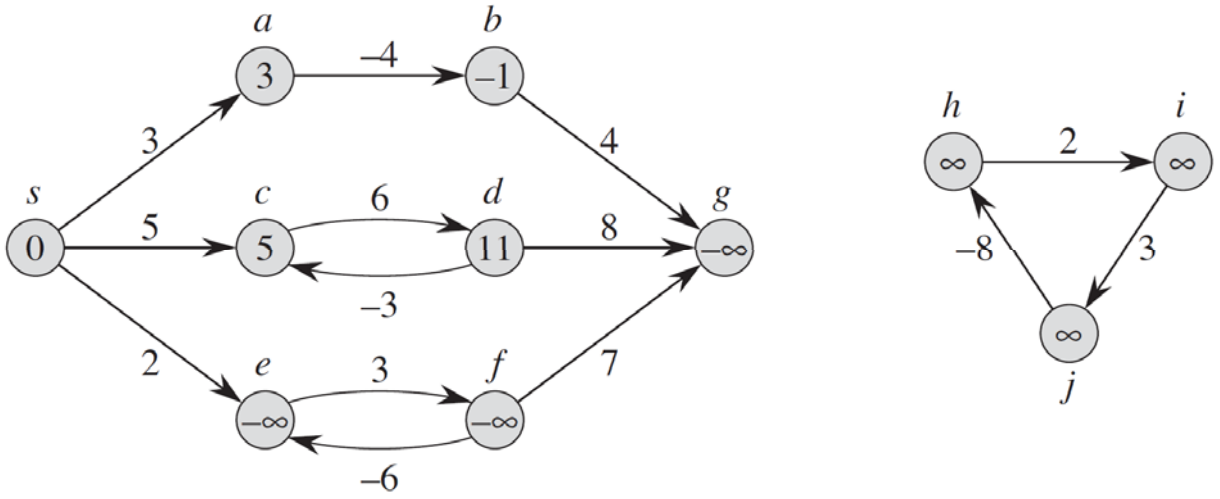


- The shortest path might **not** be unique.
- When we look at shortest paths from one vertex to all other vertices, the shortest paths are organized as a **tree**.
- The weights can represent
  - time, cost, penalties, loss.

Variants

- **Single-destination shortest-paths problem:** Find shortest paths to a given destination vertex.
  - By reversing the direction of each edge in the graph, we can reduce this problem to a single-source problem.
- **Single-pair shortest-paths problem:** Find shortest path from u to v for given vertices u and v.
  - All know algorithms have the same running time as the single-source algorithms.
- **All-pairs shortest-paths problem:** Find shortest path from u to v for all $u, v \in V$. We'll see algorithms for all-pairs in the next chapter.

Negative-weight edges

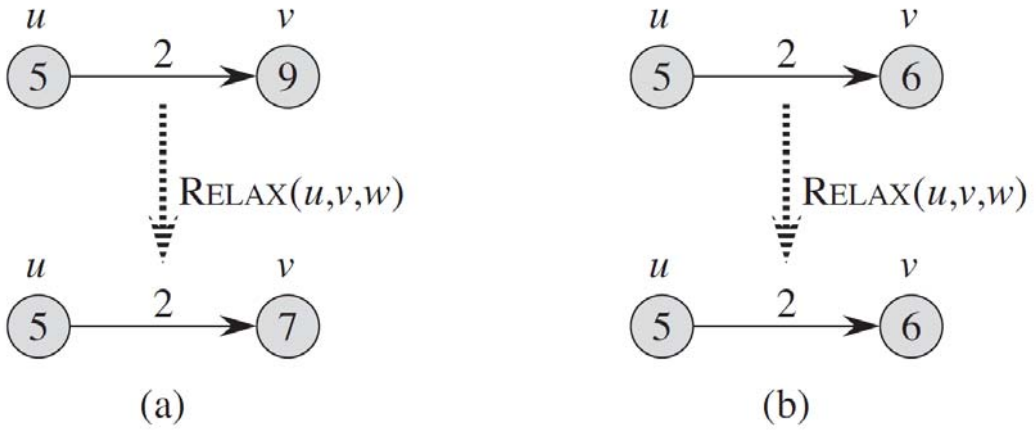


- If G contains no negative-weight cycles reachable from s, then $\delta(s,v)$ is well-defined for all $v \in V$.
- If there is a negative-weight cycle on some path from s to v, we define $\delta(s,v) = -\infty$.

Output of single-source shortest-path algorithm

- For each vertex $v \in V$:
  - $d[v] = \delta(s, v)$.
  - Initially, $d[v] = \infty$.
  - Reduces as algorithms progress.
  - But always maintain $d[v] \geq \delta(s, v)$.
- $\pi[v]$: the predecessor of v on a shortest path from s.
  - If no predecessor, $\pi[v] = \text{NIL}$.
  - π induces a tree ➔ shortest-path tree.
- **Predecessor subgraph:** $G_\pi = (V_\pi, E_\pi)$
  - $V_\pi = \{v \in V : \pi[v] \neq \text{NIL}\} \cup \{s\}$
  - $E_\pi = \{(\pi[v], v) : v \in V_\pi - \{s\}\}$

Initialization & Relaxation



- All algorithm start with INITIALIZE-SINGLE-SOURCE and then repeatedly decrease $d[v]$ until $d[v] \geq \delta(s, v)$.

INITIALIZE-SINGLE-SOURCE(G, s)
1. **for** each vertex $u \in V[G]$
2. $d[u] \leftarrow \infty$
3. $\pi[u] \leftarrow$ NIL
4. $d[s] \leftarrow 0$

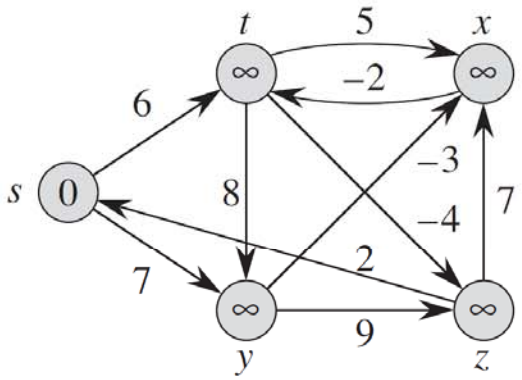
RELAX(u, v, w)
1. **if** $d[v] > d[u] + w(u, v)$
2. $d[v] \leftarrow d[u] + w(u, v)$
3. $\pi[v] \leftarrow u$

The Bellman-Ford algorithm

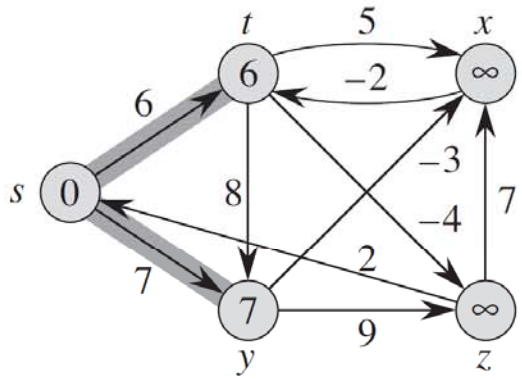
- Allows negative-weight edges.
- Computes $d[v]$ and $\pi[v]$ for all $u \in V$.
- Returns **TRUE** if no negative-weight cycles reachable from s, **FALSE** otherwise.

```
BELLMAN-FORD(G, w, s)
1.  INITIALIZE-SINGLE-SOURCE(G, s)
2.  for i = 1 to n − 1
3.      for each edge (u, v) ∈ E
4.          RELAX(u, v, w)
5.  for each edge (u, v) ∈ E
6.      if d[v] > d[u] + w(u, v)
7.          return FALSE
8.  return TRUE
```

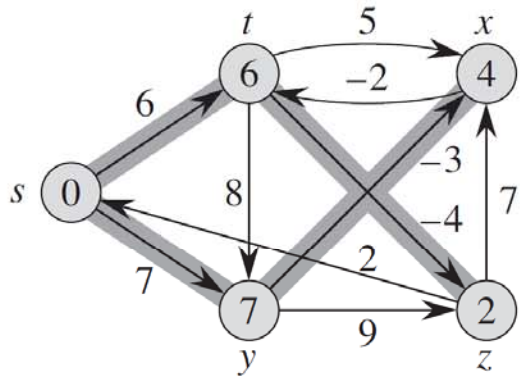
- The first for loop relaxes all edges $n - 1$ times.
- Time: $\Theta(nm)$.


t
5
x
∞
−2
∞
6
−3
s
0
8
−4
7
2
7
∞
∞
9
y
z

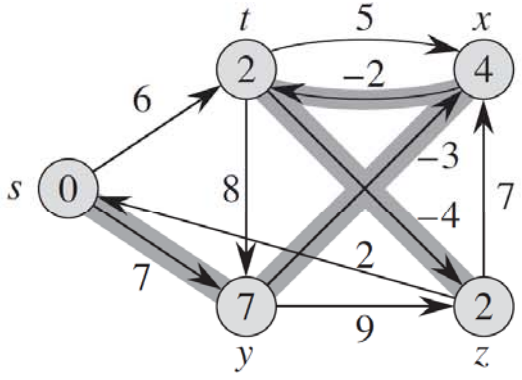
(a)


t
5
x
6
−2
∞
6
−3
s
0
8
−4
7
2
7
7
∞
9
y
z

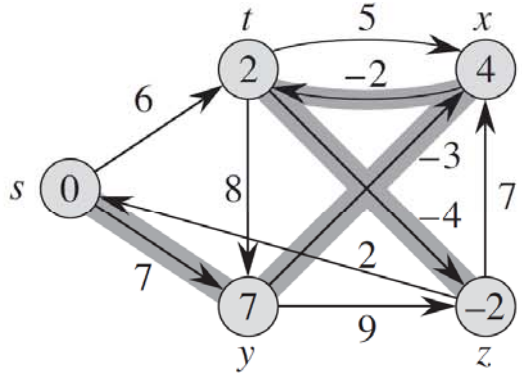
(b)


t
5
x
6
−2
4
6
−3
s
0
8
−4
7
2
7
7
2
9
y
z

(c)


t
5
x
2
−2
4
6
−3
s
0
8
−4
7
2
7
7
2
9
y
z

(d)


t
5
x
2
−2
4
6
−3
s
0
8
−4
7
2
7
7
−2
9
y
z

(e)

Outline

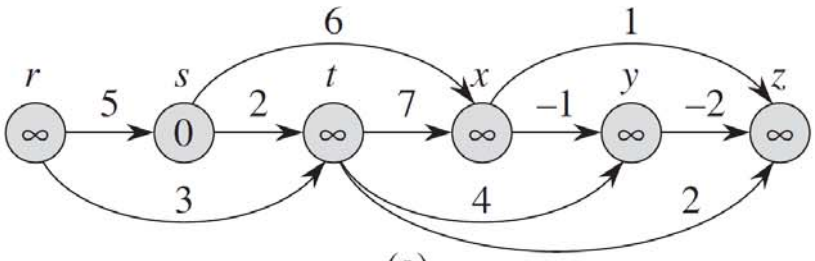
- The Bellman-Ford algorithm
- **Single-source shortest paths in directed acyclic graphs**
- Dijkstra's algorithm

Single-source shortest paths in directed acyclic graphs

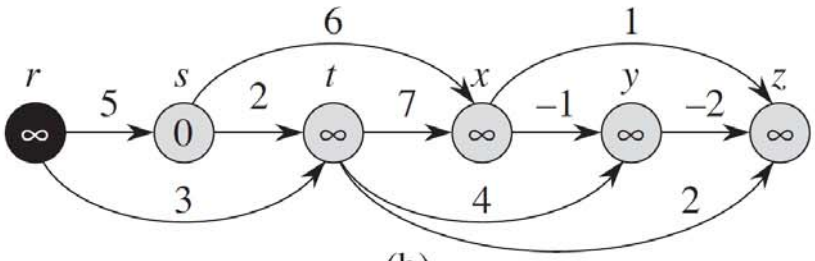
- Since G is a dag, no negative-weight cycles can exist.
- By relaxing the edges of G according to a topological sort of its vertices, we can compute shortest paths from a single source in $\Theta(n+m)$ time.

```
DAG-SHORTEST-PATHS (G, w, s)
1.    topologically sort the vertices of G
2.    INITIALIZE-SINGLE-SOURCE(G, s)
3.    for each vertex u, taken in topologically sorted order
4.          for each vertex v ∈ Adj[u]
5.                RELAX(u, v, w)
```

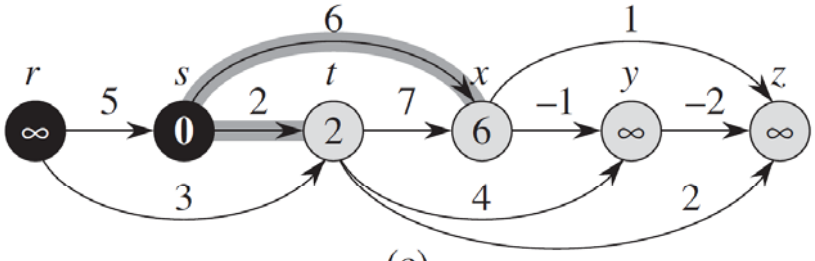
- Time: $\Theta(n+m)$.


r
s
t
x
y
z
∞
0
∞
∞
∞
∞
5
2
7
−1
−2
6
1
3
4
2

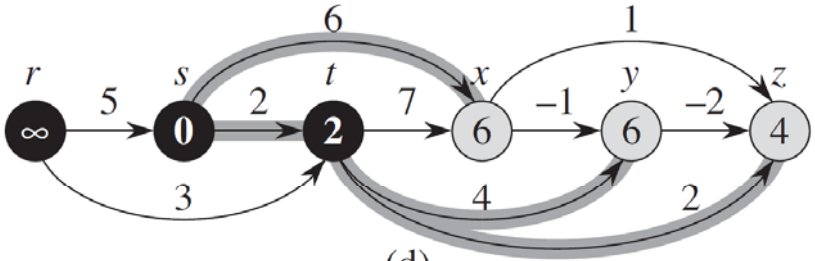
(a)


r
s
t
x
y
z
∞
0
∞
∞
∞
∞
5
2
7
−1
−2
6
1
3
4
2

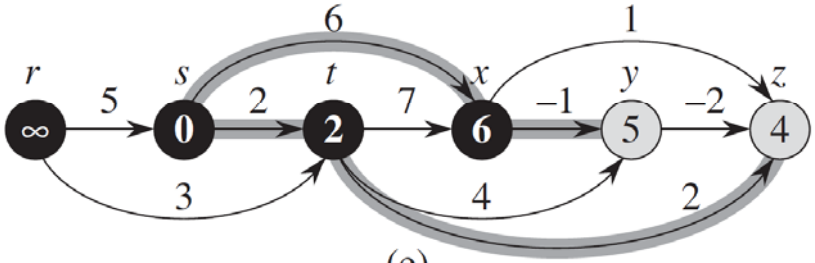
(b)


r
s
t
x
y
z
∞
0
2
6
∞
∞
5
2
7
−1
−2
6
1
3
4
2

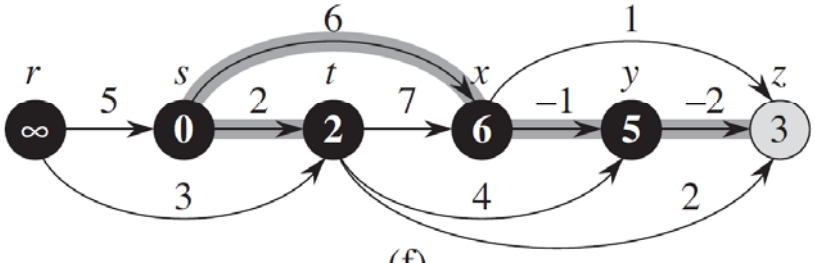
(c)


r
s
t
x
y
z
∞
0
2
6
6
4
5
2
7
−1
−2
6
1
3
4
2

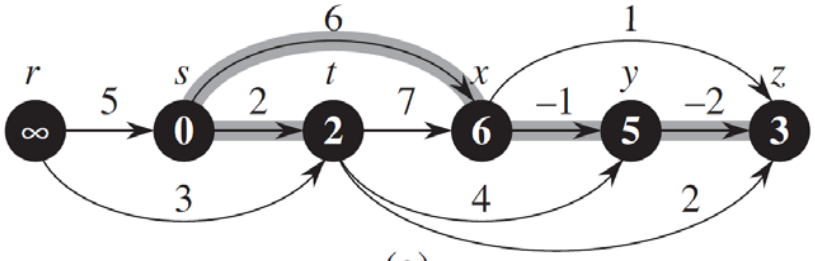
(d)


r
s
t
x
y
z
∞
0
2
6
5
4
5
2
7
−1
−2
6
1
3
4
2

(e)


r
s
t
x
y
z
∞
0
2
6
5
3
5
2
7
−1
−2
6
1
3
4
2

(f)


r
s
t
x
y
z
∞
0
2
6
5
3
5
2
7
−1
−2
6
1
3
4
2

(g)

Outline

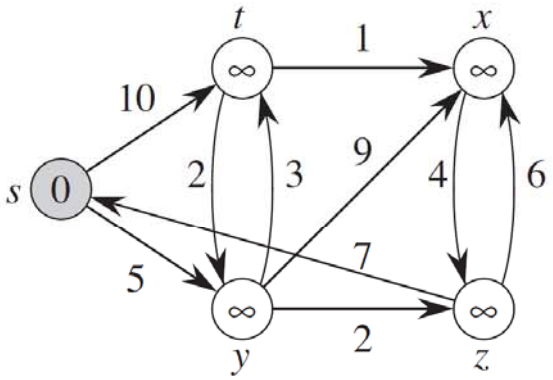
- The Bellman-Ford algorithm
- Single-source shortest paths in directed acyclic graphs
- **Dijkstra's algorithm**

Dijkstra's algorithm

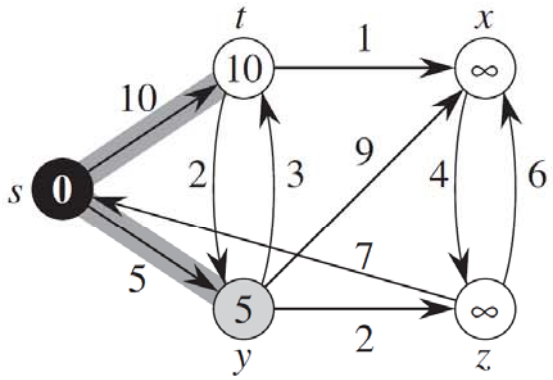
- **No** negative-weight edges.
- Essentially a weighted version of breadth-first search.
  - Instead of a FIFO queue, uses a priority queue.
  - Keys are shortest-path weights ($d[v]$).
- Have two sets of vertices:
  - S = vertices whose final shortest-path weights are determined.
  - Q = priority queue = $V - S$.

INITIALIZE-SINGLE-SOURCE(G, s)
1. **for** each vertex $u \in V[G]$
2. $d[u] \leftarrow \infty$
3. $\pi[u] \leftarrow$ NIL
4. $d[s] \leftarrow 0$

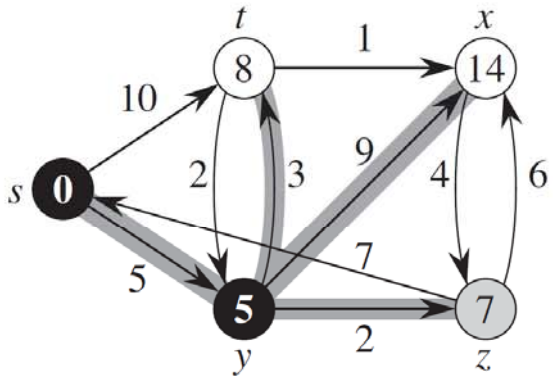
RELAX(u, v, w)
1. **if** $d[v] > d[u] + w(u, v)$
2. $d[v] \leftarrow d[u] + w(u, v)$
3. $\pi[v] \leftarrow u$


t
x
1
∞
∞
10
9
s
0
2
3
4
6
7
5
∞
∞
y
2
z

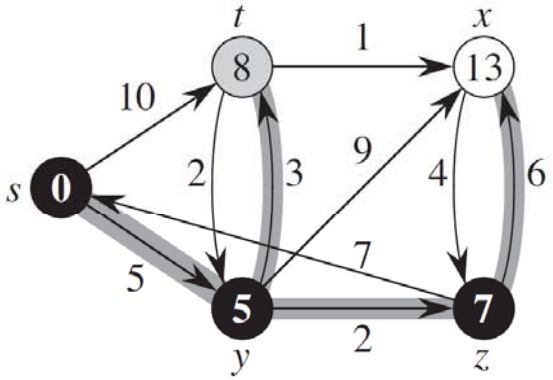
(a)


t
x
1
10
∞
10
9
s
0
2
3
4
6
7
5
5
∞
y
2
z

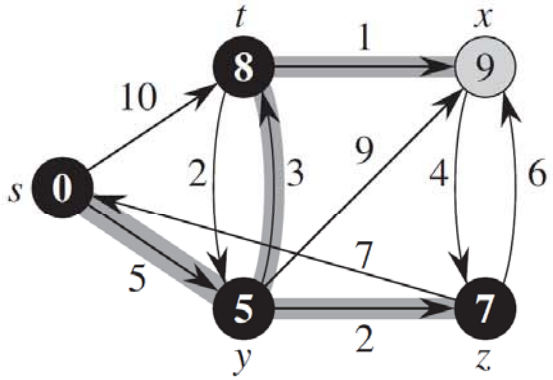
(b)


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

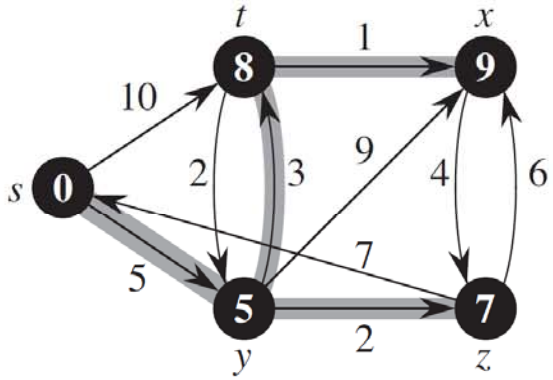
(c)


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

(d)


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

(e)


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

(f)

Dijkstra's algorithm

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

	Binary heap	Fibonacci heap
INITIALIZE-SINGLE-SOURCE	$O(n)$	$O(n)$
EXTRACT-MIN	$O(n \lg n)$	$O(n)$
RELAX	$O(m \lg n)$	$O(m)$
Total:	$O(m \lg n)$	$O(m + n \lg n)$

- Looks a lot like Prim's algorithm, but computing $d[v]$, and using shortest-path weights as keys.
- Dijkstra's algorithm can be viewed as greedy, since it always chooses the "lightest" vertex in $V - S$ to add to S.