Algorithms
Chapter 24
Single-Source Shortest Paths

給定一個點到其他點的最短距離

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** negative-weight edge：可
  $\Theta(nm)$
- Single-source shortest paths in directed acyclic graphs
  沒"cycle"有向圖的最短距離
  $\Theta(n+m)$
- Dijkstra's algorithm
  negative-weight edge：不可
  $O(m\lg n)$

給定一個點到其他點的最短距離

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:

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

路徑長度：路徑長度上各段距離的和

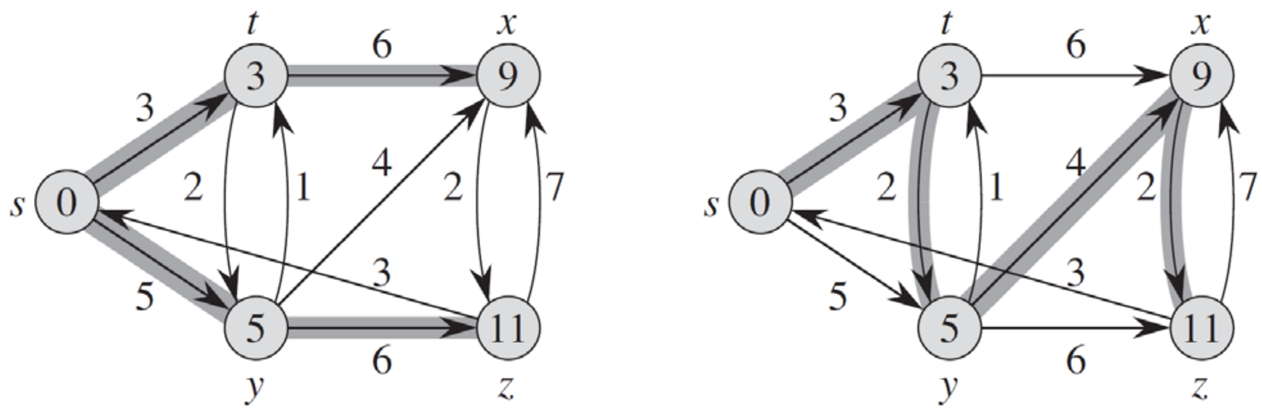
- The shortest path weight $\delta(u,v)$ from u to v is

$\delta(u,v)$: u到v最短距離的值

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

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

An example



和MST相同，最短路徑，非唯一

- The shortest path might **not** be unique.

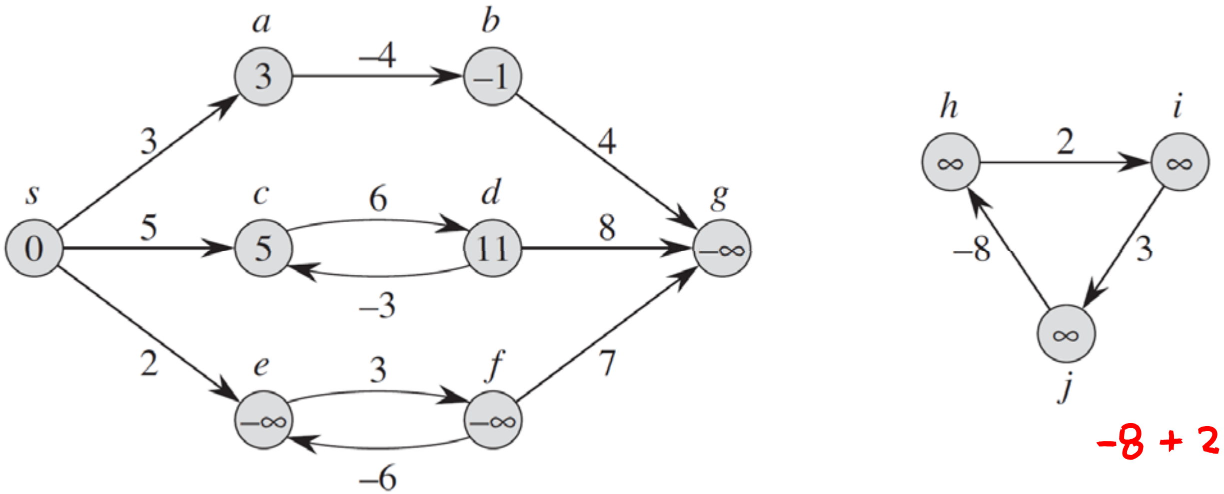
到各點的路徑會形成一個tree

- 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 weight可以表示時間、花費、損失
  - time, cost, penalties, loss.

Variants 一些相關問題

- **Single-destination shortest-paths problem:** Find shortest paths to a given destination vertex. 其它點到給定一點的最短距離
  - By reversing the direction of each edge in the graph, we can reduce this problem to a single-source problem.
    作法：將每一個邊的方向"reverse"，然後跑 single-source 演算法
- **Single-pair shortest-paths problem:** Find shortest path from *u* to *v* for given vertices *u* and *v*. 給定2點，u到v的最短距離
  - All know algorithms have the same running time as the single-source algorithms. 目前知道的方法都和 single-source 一樣快
- **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. 全部任意二點的最短距：ch 25

Negative-weight edges 花費可能為負: 下坡



$-8+2+3=-3<0$

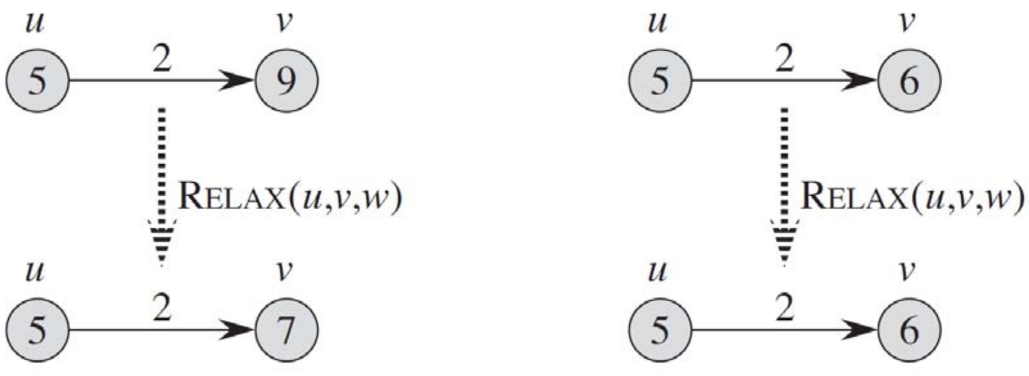
- If G contains no negative-weight cycles reachable from s, then $\delta(s,v)$ is well-defined for all $v \in V$. 如果沒有 negative-weight cycle $\Rightarrow \delta(u,v)$ 是定義良好的，無歧義
- If there is a negative-weight cycle on some path from s to v, we define $\delta(s,v) = -\infty$. 如果有，則定義 $\delta(u,v) = -\infty$

Output of single-source shortest-path algorithm

- For each vertex $v \in V$: d[v]: 記錄目前s到v的最短距離
  - $d[v] = \delta(s, v)$. 一開始設定 d[v] = ∞
  - Initially, $d[v] = \infty$.
  - Reduces as algorithms progress.
  - But always maintain $d[v] \geq \delta(s, v)$. 過程中保持 d[v] ≥ δ(s,v)
- $\pi[v]$: the predecessor of v on a shortest path from s.
  - If no predecessor, $\pi[v]$ = NIL. π[v]: s到v路徑上，v的前一個點
  - π induces a tree ➔ shortest-path tree. 一開始設定 π[v] = NULL
- **Predecessor subgraph:** $G_\pi = (V_\pi, E_\pi)$
  - $V_\pi = \{v \in V : \pi[v] \neq \text{NIL}\} \cup \{s\}$ 所有 (π[v], v) edges 会形成一棵 tree
  - $E_\pi = \{(\pi[v], v) : v \in V_\pi - \{s\}\}$

Initialization & Relaxation

Relax：如果經過 u 距離更短
① 縮短最短距
② 更新前一点



(a) (b)

- 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)

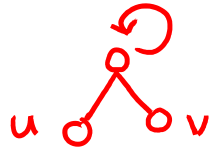
```
for each vertex u ∈ V[G]
    d[u] ← ∞
    π[u] ← NIL
d[s] ← 0
```

RELAX(u, v, w)

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

The Bellman-Ford algorithm

- Allows negative-weight edges. 可以有負"edge"
- Computes $d[v]$ and $\pi[v]$ for all $u \in V$. 記錄目前s到v的最短距離和，v的前一個点
- Returns **TRUE** if no negative-weight cycles reachable from s, **FALSE** otherwise.



path超过 n-1 边，点会重覆，删除更小
⇒最短path最多有 n-1 边

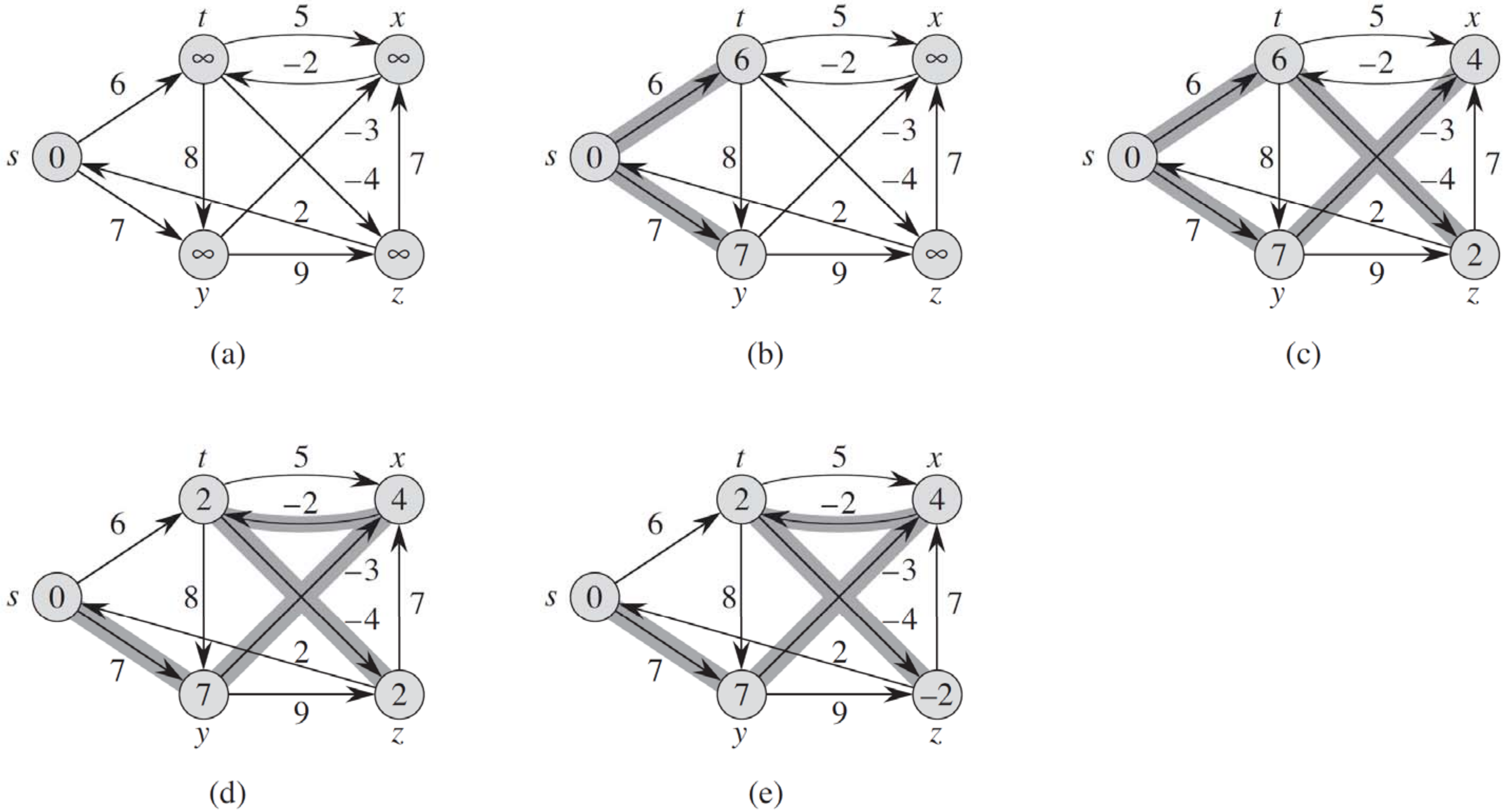
```
BELLMAN-FORD(G, w, s)
    INITIALIZE-SINGLE-SOURCE(G, s)
    for i = 1 to n − 1
        for each edge (u, v) ∈ E
            RELAX(u, v, w)
    for each edge (u, v) ∈ E
        if d[v] > d[u] + w(u, v)
            return FALSE
    return TRUE
```

edge relax n-1 次

檢測是否有負cycle

∵ s到v最多經过 n-1 边 relax n-1 次後不会再減少，除非有負"cycle"

- The first for loop relaxes all edges $n-1$ times.
  將所有edge，relax n-1 次
- Time: $\Theta(nm)$.



s
t
x
y
z
0
∞
6
7
2
4
−2
5
8
−3
−4
9
(a)
(b)
(c)
(d)
(e)

Outline

- The Bellman-Ford algorithm
- **Single-source shortest paths in directed acyclic graphs**
  沒"cycle"有向圖的最短距離
- Dijkstra's algorithm

dag：沒有"cycle"的有向圖

Single-source shortest paths in directed acyclic graphs

- Since G is a dag, no negative-weight cycles can exist.

  因為是 dag，所以沒有負"cycle"

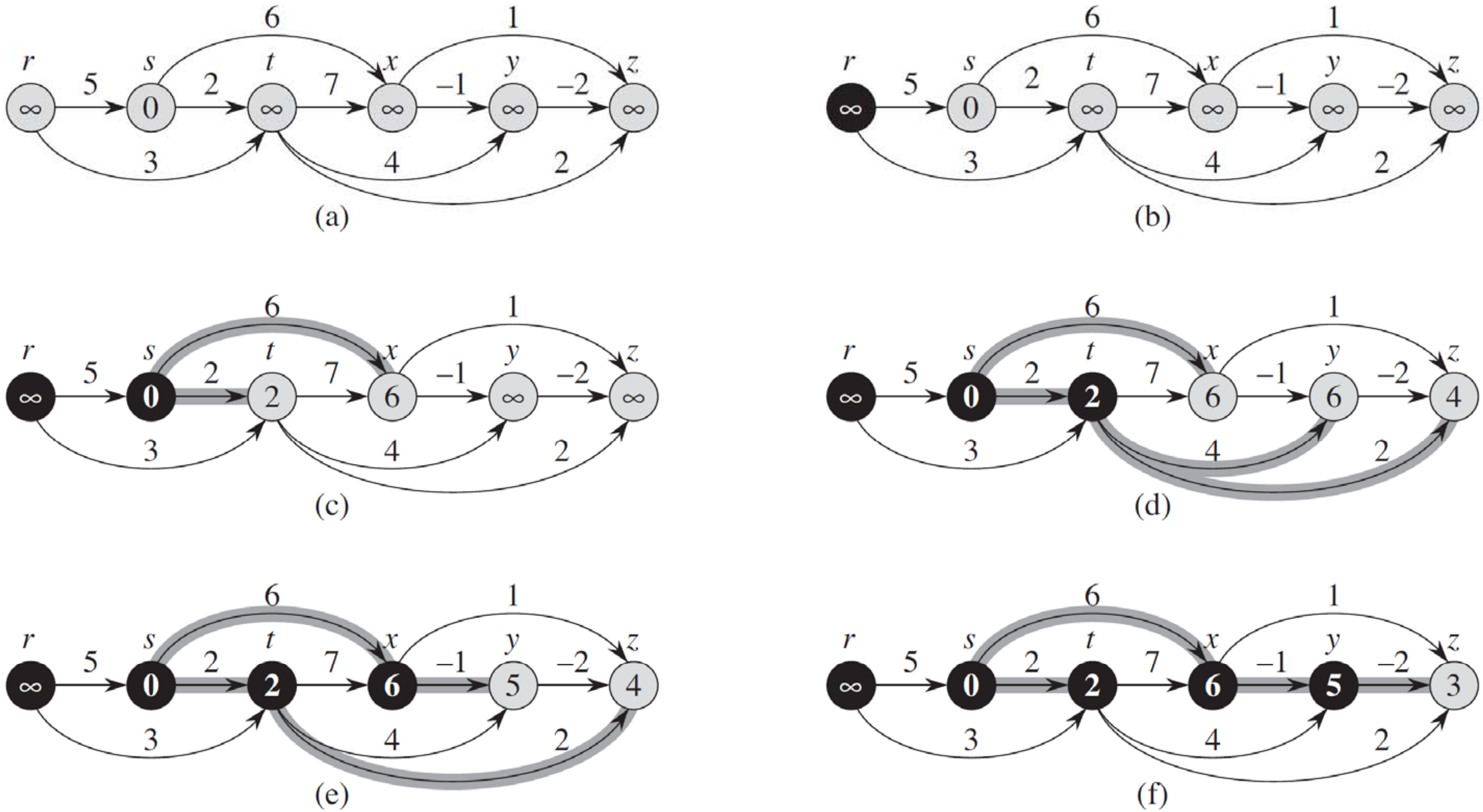
- By relaxing the edges of G according to a topological sort of its vertices, we can compute shortest paths from a single source in $\Theta(n+m)$ time. 利用 topological sort，將 time 由 $\Theta(nm) \rightarrow \Theta(n+m)$

DAG-SHORTEST-PATHS (G, w, s)

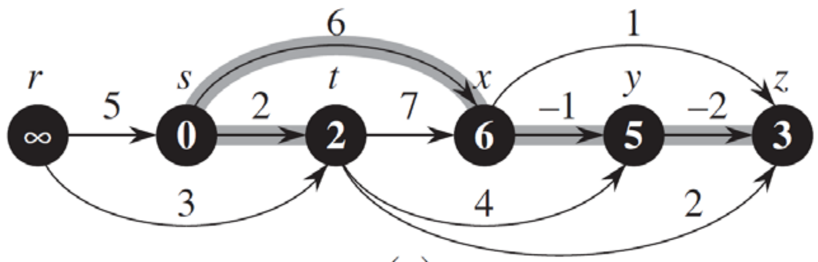
```
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
 for each vertex u, taken in topologically sorted order
     for each vertex v ∈ Adj[u]
         RELAX(u, v, w)
```

使用 topological sort 當 relax 的順序

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



(a) (b) (c) (d) (e) (f)



(g)

Dag: ① 排序

② 由前到後 relax 鄰居

Outline

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

negative-weight edge : 不可

Dijkstra's algorithm

- **No** negative-weight edges. negative-weight edge：不可
- Essentially a weighted version of breadth-first search.
  - Instead of a FIFO queue, uses a priority queue. 利用 priority queue 選最小
  - Keys are shortest-path weights ($d[v]$). 使用 d[v] 当 key 值
- Have two sets of vertices: 過程中維護兩個集合 S 和 Q
  - S = vertices whose final shortest-path weights are determined.
  - Q = priority queue = $V - S$.

S = 最短路徑已經決定的点集合
Q = V - S
= 在 priority queue 中的点集合

```
INITIALIZE-SINGLE-SOURCE(G, s)
    for each vertex u ∈ V[G]
        d[u] ← ∞
        π[u] ← NIL
    d[s] ← 0
```

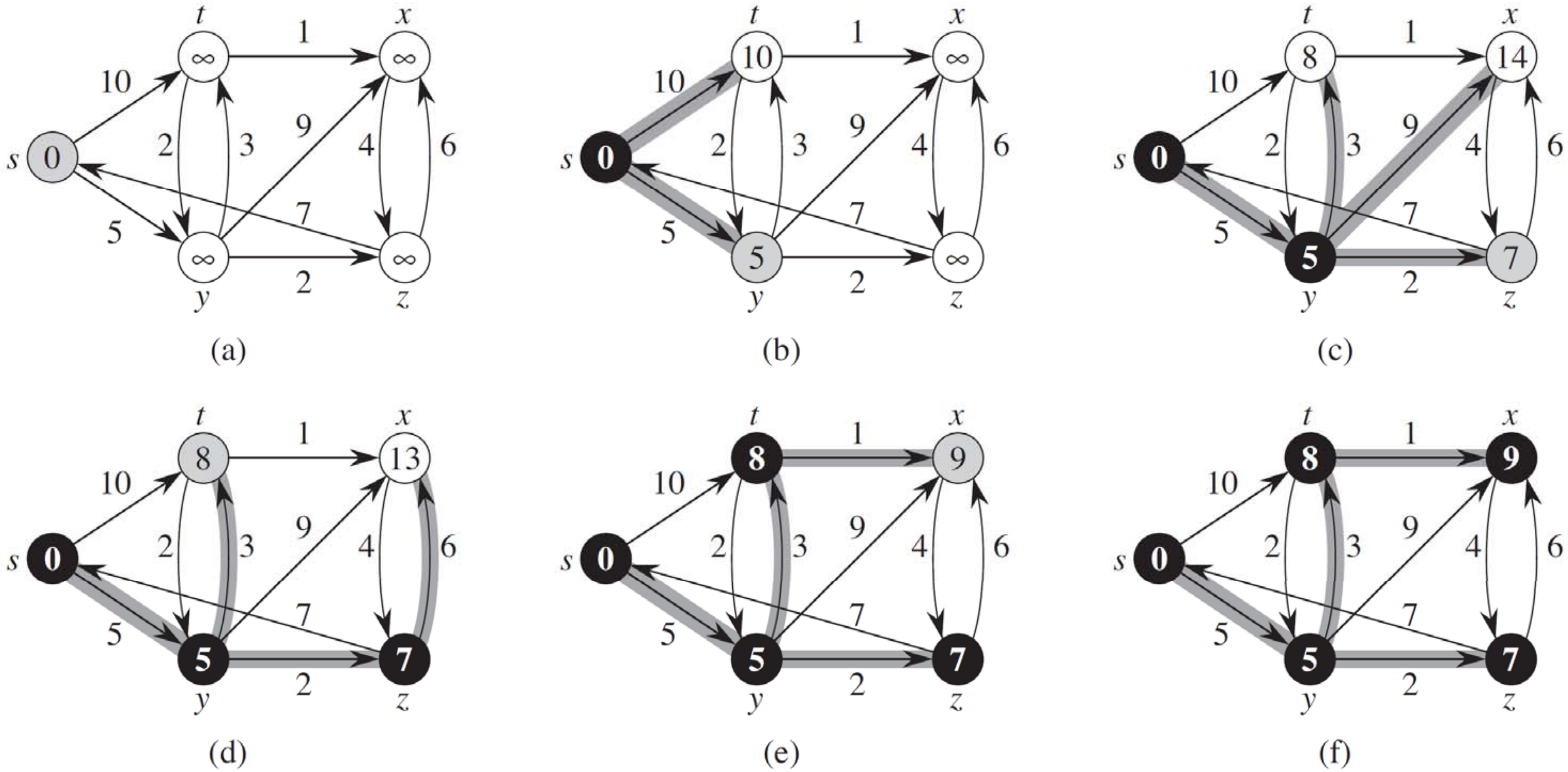
```
RELAX(u, v, w)
    if d[v] > d[u] + w(u, v)
        d[v] ← d[u] + w(u, v)
        π[v] ← u
```

①為一貪婪演算法

②以出發點為核心擴大

③動作:(a)選最小

(b)更新鄰居資訊



Prim：更新 w(u,v)，v 到集合 S 的最短距離

Dijkstra：更新 d[v]，s 到 v 的最短距離

S = 與起始点相連的所有点

s = 出發点

Dijkstra’s algorithm

使用不同 heap，時間也不同

DIJKSTRA (G, w, s)

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

	Binary heap	Fibonacci heap
	$O(n)$	
	$O(n \lg n)$	
	$O(n)$	
	$O(m \lg n)$	$O(m)$
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.