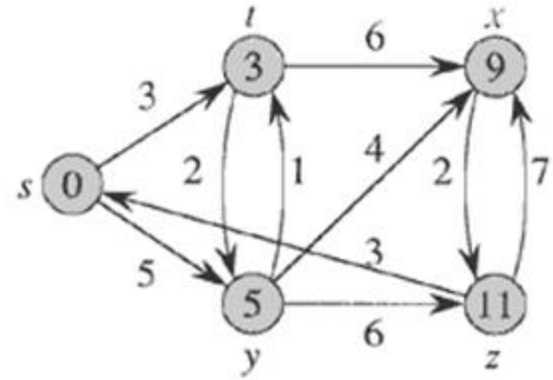
Shortest Path Algorithm

Shortest Path Problem

- Definition
  - Weighted, directed graph G=(V, E) 에 대하여, 총 weight 가 최소화 되는 path 를 찾는 문제

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



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

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

  = sum of edge weights on path p.

- *Shortest-path weight* u to v:

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

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

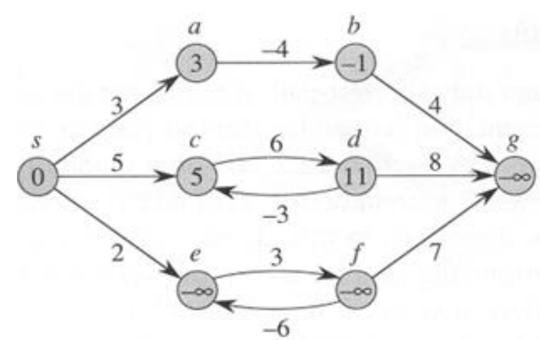
Shortest Path Problem

- Types
  1) Single-source shortest paths problem
  2) Single-destination shortest paths problem
  3) Single-pair shortest paths problem
  4) All-pairs shortest paths problem

- Solutions to single-source shortest paths problem

  (i) Negative-weight edge 가 있는 경우
  - Bellman-Ford algorithm
  - O(V E)

  (ii) Negative-weight edge 가 없는 경우
  - Dijkstra's algorithm
  - $O(V^2)$– O(V lg V + E) : 구현 방법에 따라 다름

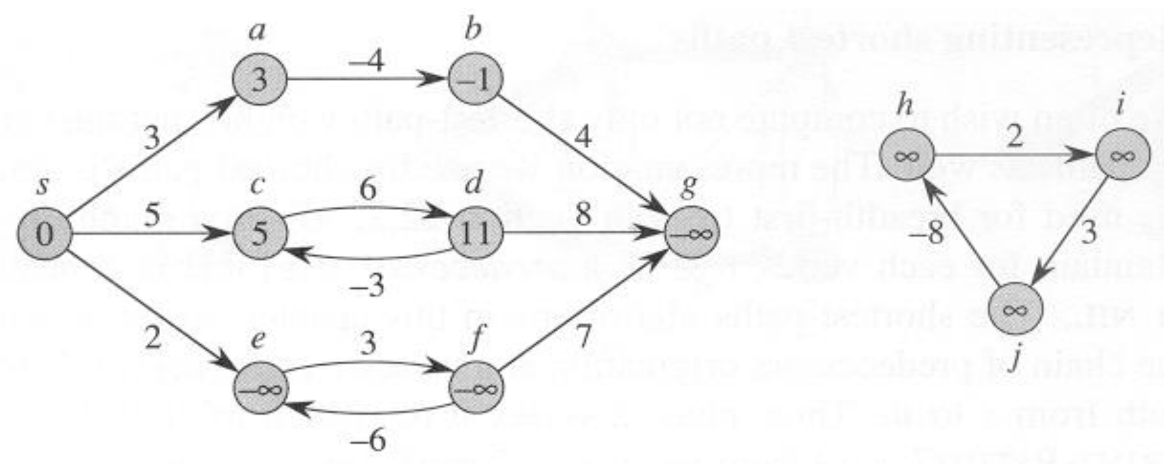


Bellman-Ford Algorithm

- 목적

  single-source shortest-paths problem 의 해를 구함

  - negative weighted edge 대응 가능
  - source 에서 도달 가능한 negative-weight cycle 존재 여부 판별 => 존재하면 해 없음.



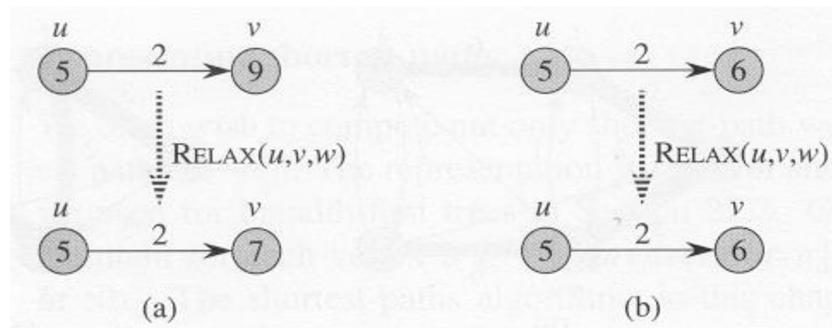
Bellman-Ford Algorithm

- 원리
  - Relaxation

    d[*v*]: s 부터 v 까지의 shortest-path estimate

    RELAX(u,v,w)

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



  - Path-relaxation property

    $p = <v_0, v_1, \ldots, v_k>$ 가 v_0 에서 v_k까지의 shortest path 이고 *p* 의

    edge 들이 (v_0, v_1) ,(v_1, v_2), …, (v_{k-1}, v_k)의 순으로 relax 되었다면,

    $d[v_k] = \delta(s, v_k)$

Bellman-Ford Algorithm

- BELLMAN-FORD(G,w,s)
  - O (V E)

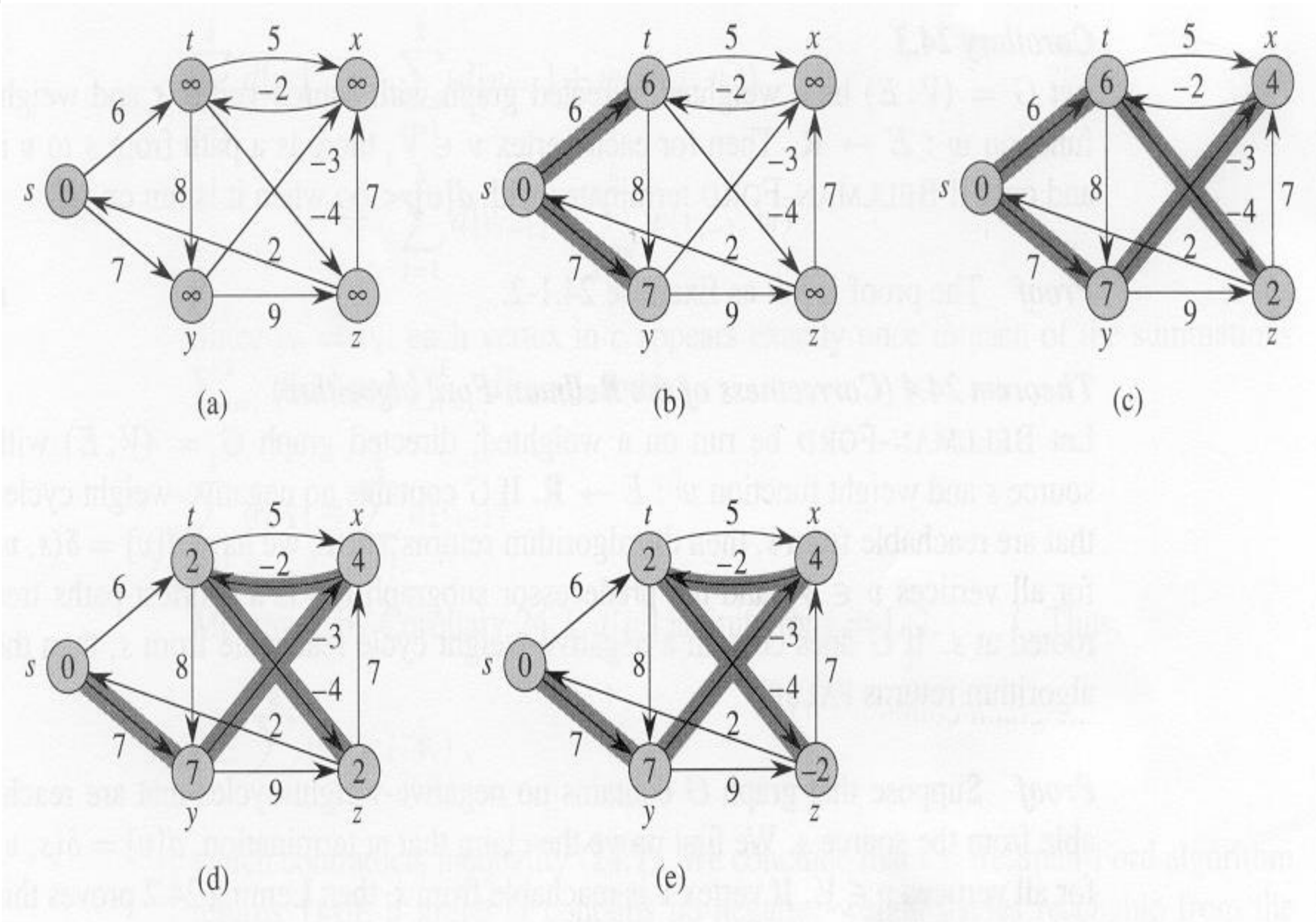
```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i ← 1 to |V[G]| − 1
     do for each edge (u, v) ∈ E[G]
            do RELAX(u, v, w)
for each edge (u, v) ∈ E[G]
     do if d[v] > d[u] + w(u, v)
           then return FALSE
return TRUE
```

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

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

Bellman-Ford Algorithm

- Operations



Bellman-Ford Algorithm

(Q)

Dijkstra’s Algorithm

- 특징
  - Single-source shortest problem
  - Nonnegative-weighted edges
  - Running time: lower than Bellman-Ford algorithm (if good implementation)
- Data
  - Graph: V, E, w(u,v)
    - adjacency-list
    - adjacency-matrix
  - d[u]: vertex u 의 shortest-path estimate
  - π[u]: vertex u 의 predecessor
  - S: source s 로 부터의 최단거리가 결정된 vertex 의 집합
  - Q: vertex 들의 minimum-priority queue, key = d

(cf) breadth first search

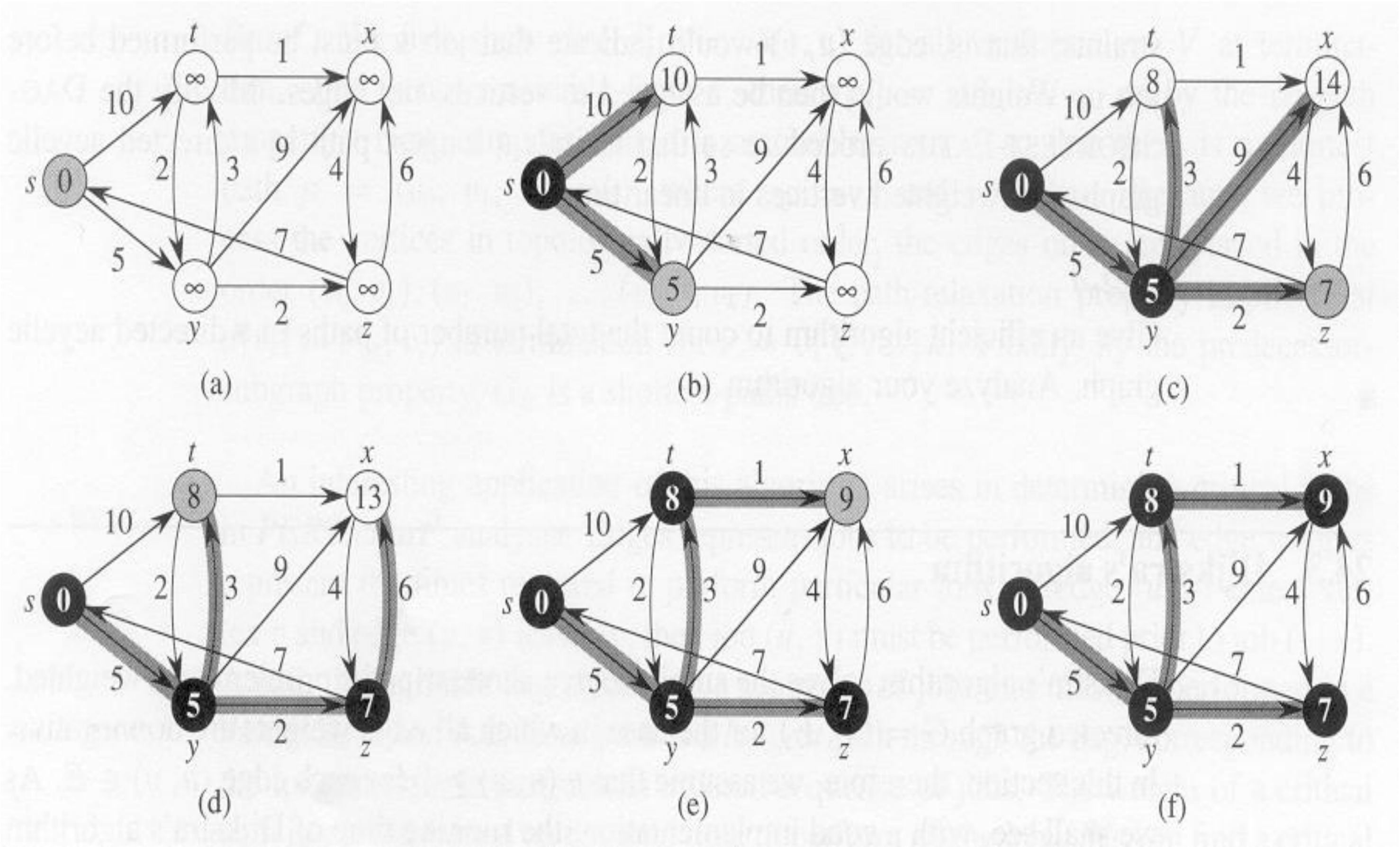
Dijkstra's Algorithm

- DIJKSTRA(G, w, s)
  - Greedy strategy
  - Optimal solution $d[u] = \delta(s, u), \forall u \in V$
  - Running time : O (V lg V + E)
    - line 4-8 : O(V)
    - EXTRACT-MIN(q): O (lg V)
    - Line 7-8: O(E)

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

Dijkstra's Algorithm

- Operations



Dijkstra's Algorithm

(Q)