24. Single-Source Shortest Paths

The problem of finding shortest paths from a source vertex s to all other vertices in the graph.

Single-Source Shortest Path Problem

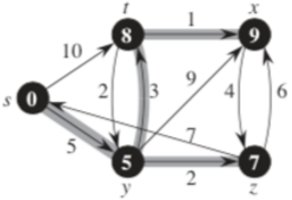
In a weighted graph $G = (E,V)$, find all $\delta(s,v)$ from a source vertex $s \in V$ to all vertices $v \in V$ where

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

The ***weight*** $w(p)$ of path $p = \langle v_0, v_1, \ldots, v_k \rangle$

We define the ***shortest-path weight*** $\delta(u, v)$ from u to v by

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



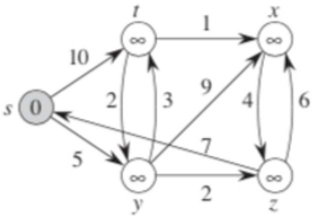
Single-Source Shortest Path Algorithms

-Bellman-Ford algorithm : works in a graph with negative weights

-Dijkstra's algorithm : works in a graph with nonnegative weights

Variants of single-source shortest paths problem

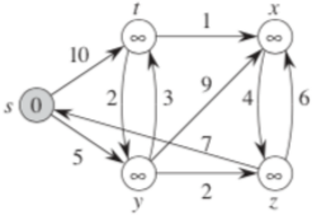
- Single-destination shortest-paths problem : edge 방향을 반대로 하고 single-source shortest-paths problem을 푼다.



- Single-pair shortest-path problem : single-source shortest-paths problem을 풀면 그 안에 해가 포함되어 있다. Single-pair shortest-path problem 만 푸는 알고리즘의 worst-case running time 은 가장 좋은 single-source shortest-paths problem 의 worst-case running time 과 점근적으로 같다.

Variants of single-source shortest paths problem (2)

- All-pairs shortest-paths problem : 모든 vertices 에 대해 single-source shortest-paths problem을 푼다. 그러나 25장의 알고리즘들 (e.g. Floyd-Warshall algorithm) 과 같이 더 효율적인 방법도 있다.



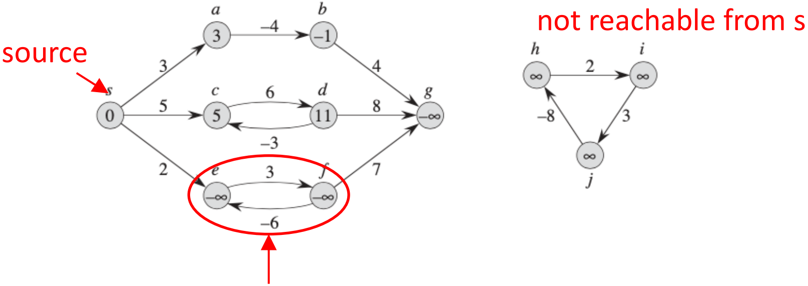
Optimal substructure of a shortest path

Lemma 24.1 (Subpaths of shortest paths are shortest paths)
Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$, let $p = \langle v_0, v_1, \ldots, v_k \rangle$ 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 $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ be the subpath of p from vertex v_i to vertex v_j. Then, p_{ij} is a shortest path from v_i to v_j.

Proof If we decompose path p into $v_0 \overset{p_{0i}}{\leadsto} v_i \overset{p_{ij}}{\leadsto} v_j \overset{p_{jk}}{\leadsto} v_k$, then we have that $w(p) = w(p_{0i}) + w(p_{ij}) + w(p_{jk})$. Now, assume that there is a path p'_{ij} from v_i to v_j with weight $w(p'_{ij}) < w(p_{ij})$. Then, $v_0 \overset{p_{0i}}{\leadsto} v_i \overset{p'_{ij}}{\leadsto} v_j \overset{p_{jk}}{\leadsto} v_k$ is a path from v_0 to v_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 v_0 to v_k. ■

Thus,
Dijkstra's algorithm : greedy algorithm
Floyd-Warshall algorithm : dynamic programming

Negative-weight edges

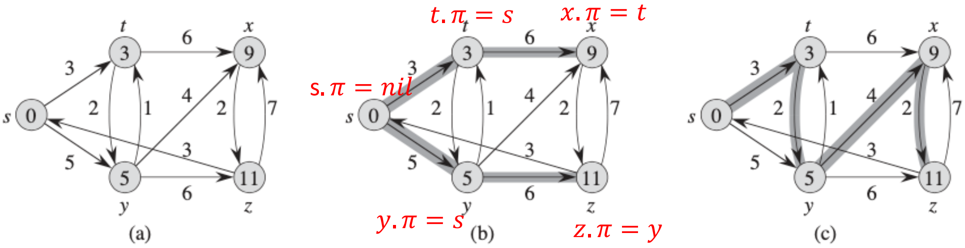


Graph G 에 negative-weight cycle 이 있으면 shortest-path problem 은 well-defined 가 아니다. s->e, s->f, s->g 때문

Bellman-Ford algorithm : works in a graph with negative weights, unless there is a negative cycle. (and detects it.)

Dijkstra's algorithm : works in a graph with nonnegative weights

Shortest-paths tree



A ***shortest-paths tree*** rooted at s is a directed subgraph $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$, such that

1. V' is the set of vertices reachable from s in G,
2. G' forms a rooted tree with root s, and
3. for all $v \in V'$, the unique simple path from s to v in G' is a shortest path from s to v in G.

predecessor subgraph defined by "$v.\pi$: v 의 predecessor in shortest-paths tree" 와 같다.

Relaxation

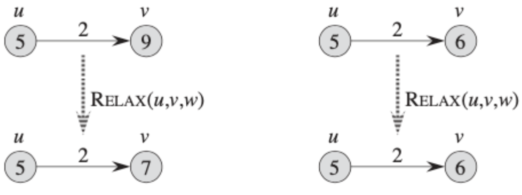
Update operation of *v.d (shortest-path estimate)*

RELAX(u, v, w) weights

if $v.d > u.d + w(u, v)$

  $v.d = u.d + w(u, v)$

  $v.\pi = u$



(a) (b)

24.1 Bellman-Ford Algorithm

- Single source shortest path algorithm
- Unlike Dijkstra's algorithm, edges can have negative weight.
- The algorithm returns false when there is a negative cycle.

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

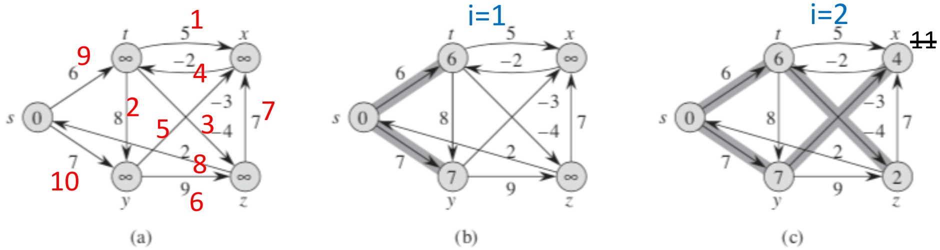
main

check for negative cycle

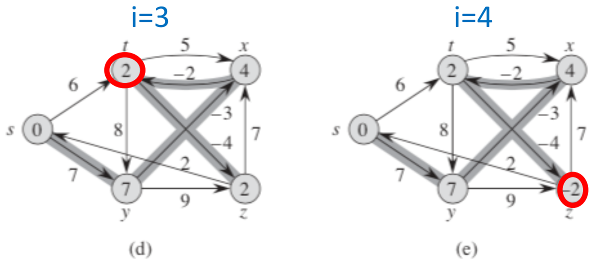
```
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex v ∈ G.V
    v.d = ∞
    v.π = NIL
s.d = 0
```

source : s

edge order : $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.



(a) (b) (c)



(d) (e)

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

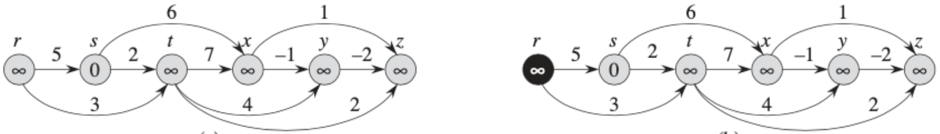
$= O(VE)$

24.2 Topological sort 를 이용한 single-source shortest path

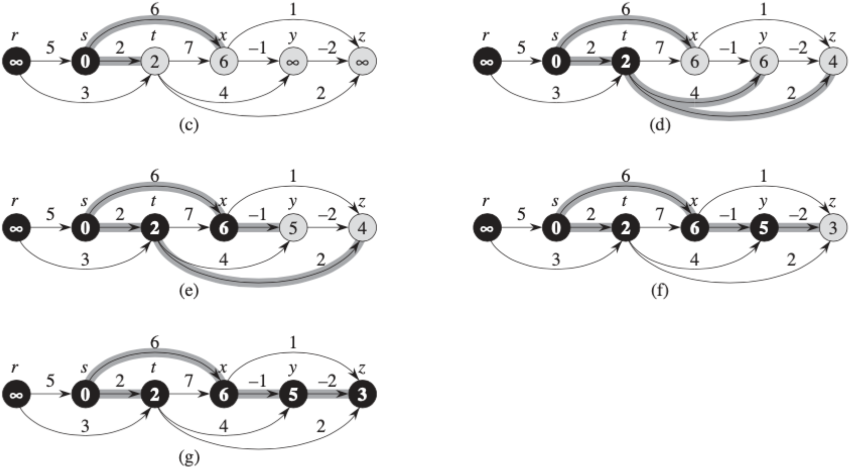
- G 가 directed acyclic graph 일 때 (cycle 이 없으므로 negative-weight cycle 도 없음) 사용 가능

```
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 ∈ G.Adj[u]
        RELAX(u, v, w)
```

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



(a) (b)



DAG-SHORTEST-PATHS(G, w, s) $= \Theta(E+V)$

topologically sort the vertices of G $= \Theta(E+V)$

INITIALIZE-SINGLE-SOURCE(G, s) $= \Theta(V)$

for each vertex u, taken in topologically sorted order $= \Theta(E)$

  for each vertex $v \in G.Adj[u]$

    RELAX(u, v, w)

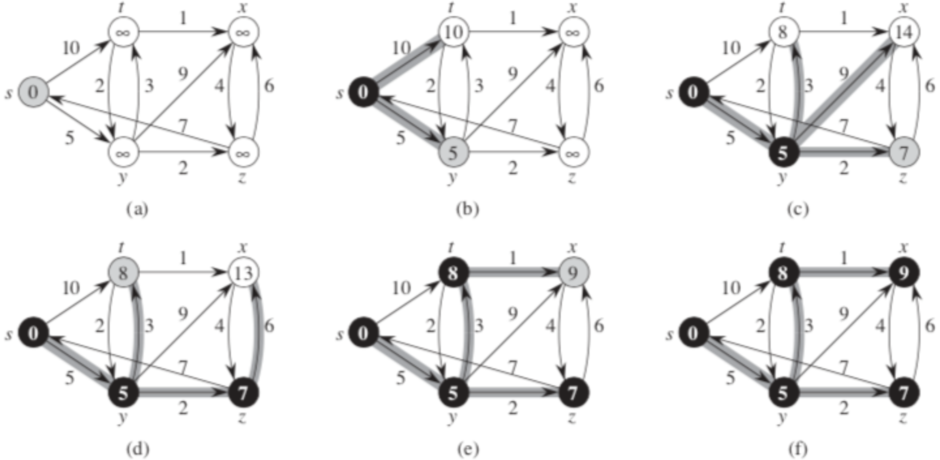
24.3 Dijkstra's Algorithm

- used when G has no negative edges.
- Greedy algorithm

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

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

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



Analysis of Dijkstra's Algorithm

DIJKSTRA(G, w, s)

INITIALIZE-SINGLE-SOURCE(G, s)
$S = \emptyset$
$Q = G.V$ *: BUILD_MIN_HEAP( ) : O(V)*
while $Q \neq \emptyset$
    $u =$ EXTRACT-MIN(Q) *: V x O(lg V)*
    $S = S \cup \{u\}$
    for each vertex $v \in G.Adj[u]$
        RELAX(u, v, w) *: DECREASE_KEY implies MIN_HEAPIFY → E x O(lg V)*

priority queue 가 binary min heap 으로 구현된 경우 running time (refer to chapter 5) = O((V+E)lg V)

Analysis of Dijkstra's Algorithm

```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = Ø
  Q = G.V                          : BUILD_MIN_HEAP( ) : O(V) : O(V)
  while Q ≠ Ø
      u = EXTRACT-MIN(Q)           : V x O(lg V) : V x O(V)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)           : DECREASE_KEY implies MIN_HEAPIFY
                                   → E x O(lg V) : E x O(1)
```

Q가 linear array 로 구현된 경우 running time $= O(V^2+E)$

priority queue 가 binary min heap 으로 구현된 경우 running time (refer to chapter 5) $= O((V+E)\lg V)$