Graph Algorithms

This lecture is based on the slides of 한빛미디어㈜

More Graph Algorithms

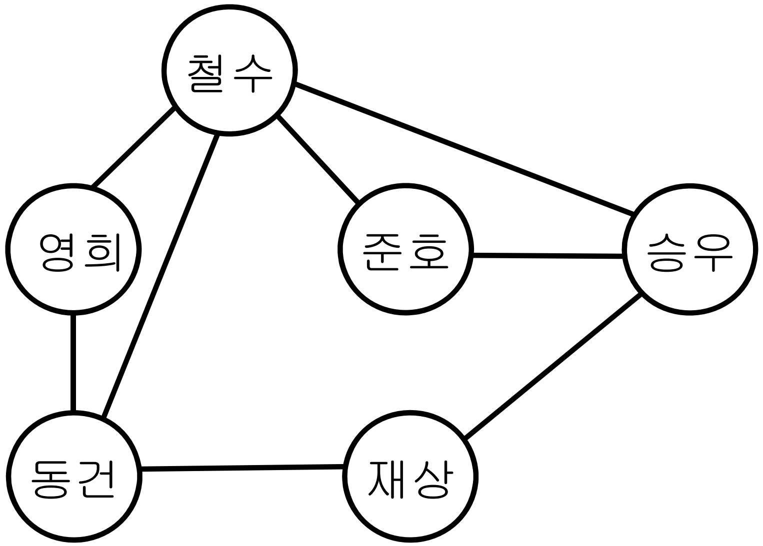
- Reminding Fundamentals of Graph Algorithms
- Bellman-Ford Algorithm
- Topological Sorting
- Strongly Connected Graph Detection

Fundamentals of Graph Algorithms (Skip..)

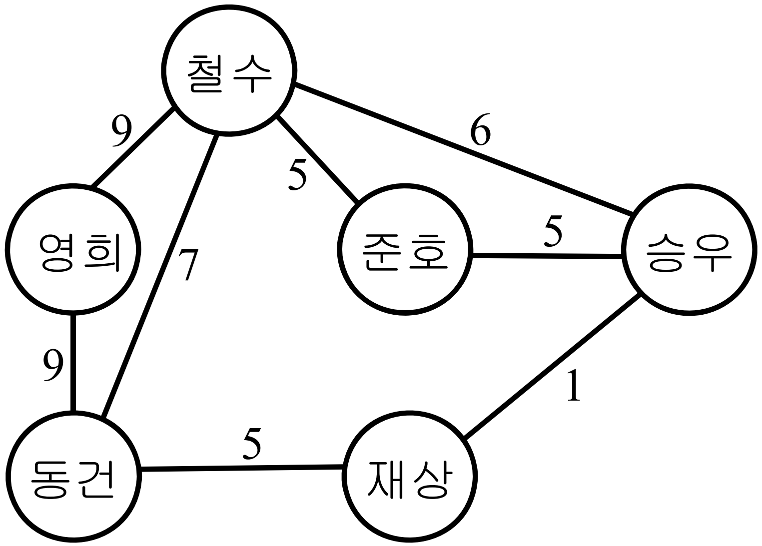
Graph

- 현상이나 사물을 정점vertex과 간선edge으로 표현한 것
- Graph $G = (V, E)$
  - V: 정점 집합
  - E: 간선 집합
- 두 정점이 간선으로 연결되어 있으면 인접하다고 한다
  - 인접 = adjacent
  - 간선은 두 정점의 관계를 나타낸다

그래프의 예

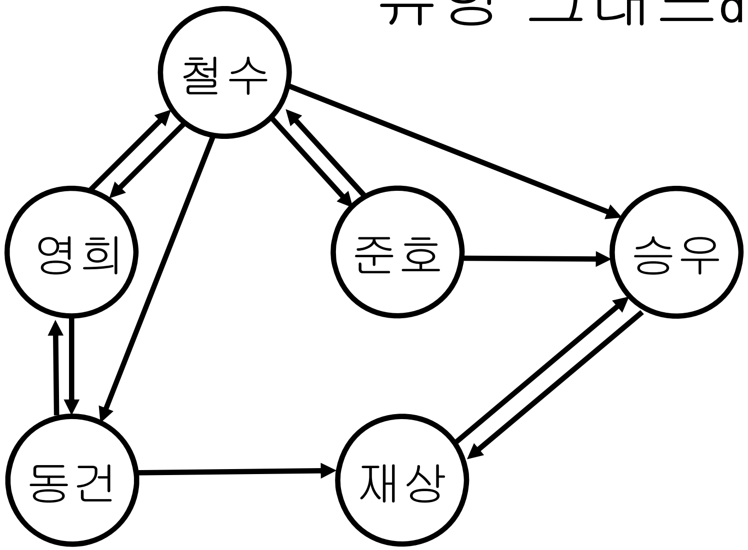


사람들간의 친분 관계를 나타낸 그래프

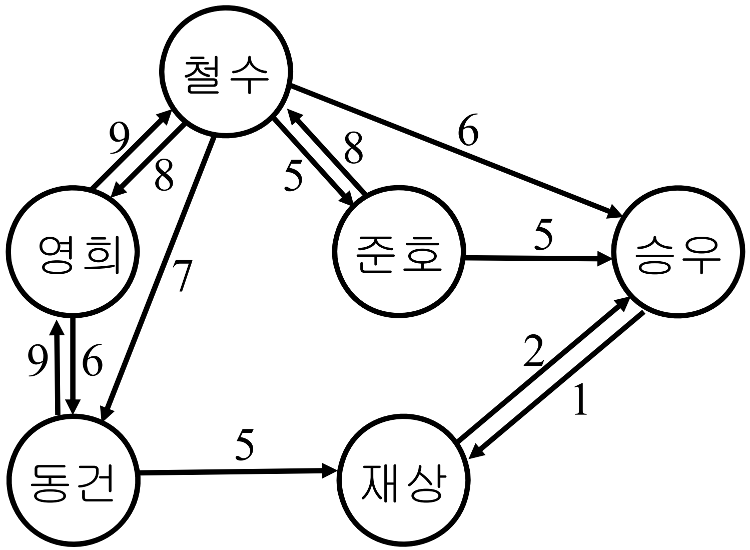


친밀도를 가중치로 나타낸 친분관계 그래프

유향 그래프directed graph=digraph



방향을 고려한 친분관계 그래프

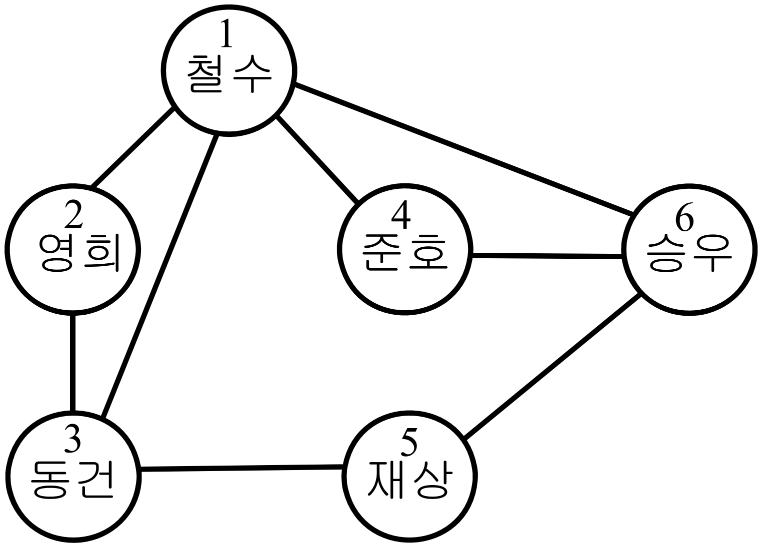


가중치를 가진 유향 그래프

Graph의 표현 1: Adjacency Matrix

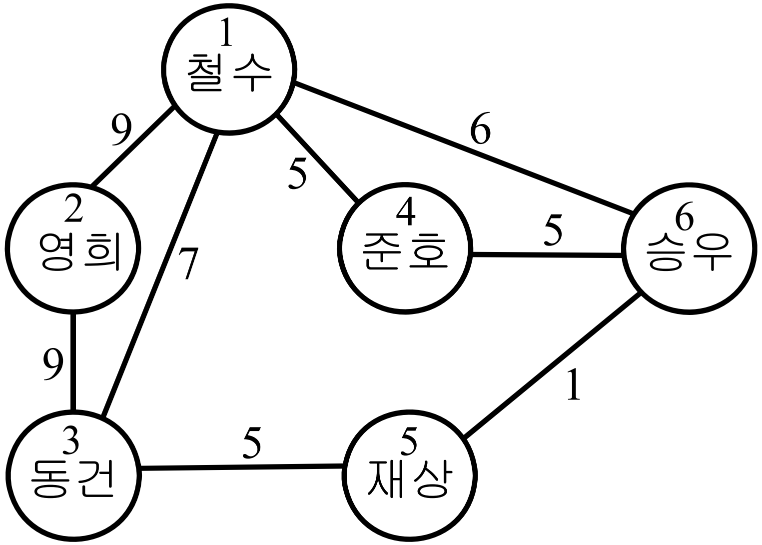
N: 정점의 총 수

- Adjacency matrix
  - N x N 행렬로 표현
    - 원소 $(i, j) = 1$: 정점 i 와 정점 j 사이에 간선이 있음
    - 원소 $(i, j) = 0$: 정점 i 와 정점 j 사이에 간선이 없음
  - 유향 그래프의 경우
    - 원소 (i, j)는 정점 i 로부터 정점 j 로 연결되는 간선이 있는지를 나타냄
  - 가중치 있는 그래프의 경우
    - 원소 (i, j)는 1 대신에 가중치를 가짐



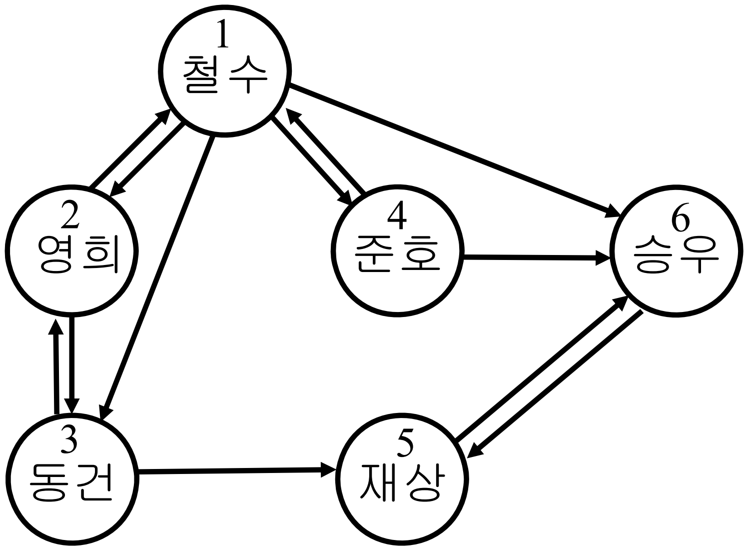
	1	2	3	4	5	6
1	0	1	1	1	0	1
2	1	0	1	0	0	0
3	1	1	0	0	1	0
4	1	0	0	0	0	1
5	0	0	1	0	0	1
6	1	0	0	1	1	0

무향 그래프의 예



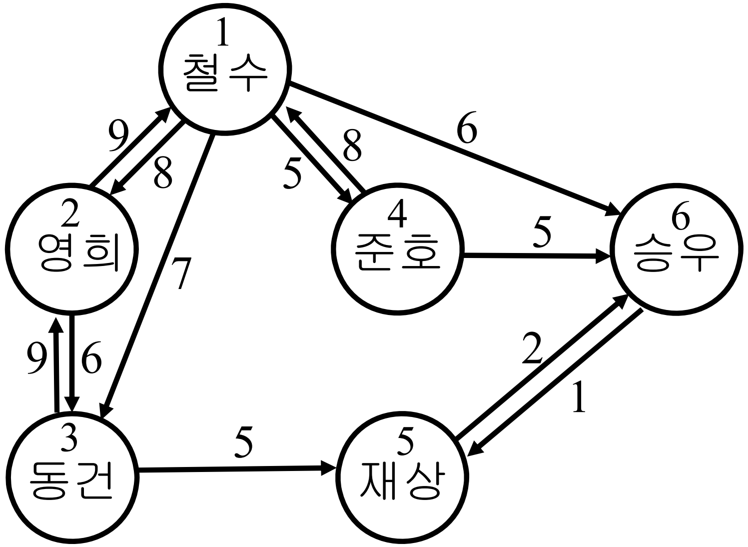
	1	2	3	4	5	6
1	0	9	7	5	0	6
2	9	0	9	0	0	0
3	7	9	0	0	5	0
4	5	0	0	0	0	5
5	0	0	5	0	0	1
6	6	0	0	5	1	0

가중치 있는 무향 그래프의 예



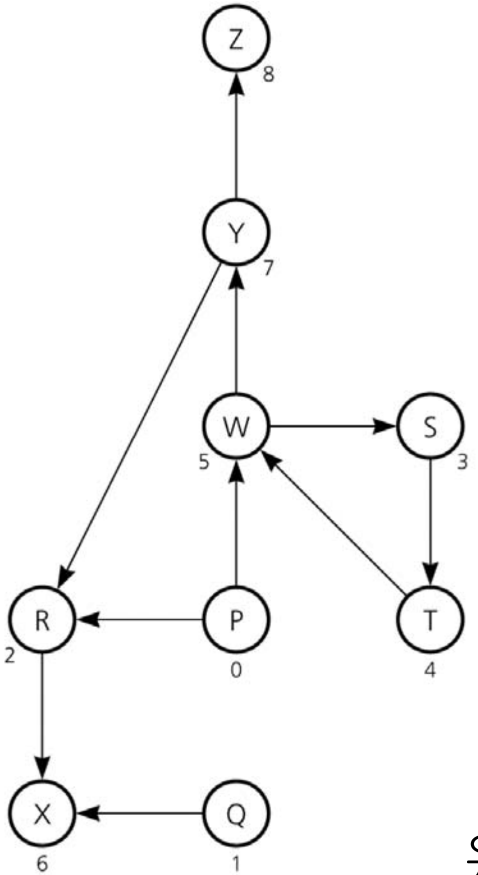
	1	2	3	4	5	6
1	0	1	1	1	0	1
2	1	0	1	0	0	0
3	0	1	0	0	1	0
4	1	0	0	0	0	1
5	0	0	0	0	0	1
6	0	0	0	0	1	0

유향 그래프의 예



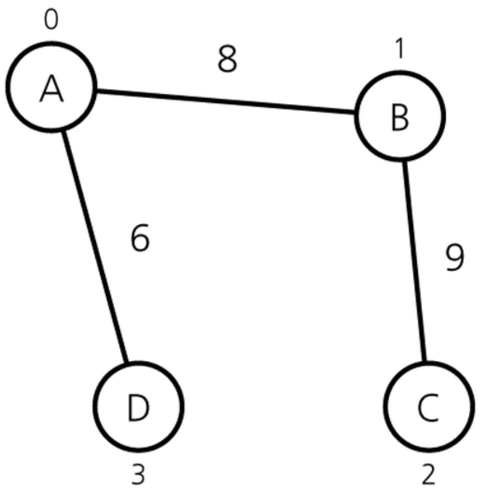
	1	2	3	4	5	6
1	0	8	7	5	0	6
2	9	0	6	0	0	0
3	0	9	0	0	5	0
4	8	0	0	0	0	5
5	0	0	0	0	0	2
6	0	0	0	0	1	0

가중치 있는 유향 그래프의 예



		0	1	2	3	4	5	6	7	8
		P	Q	R	S	T	W	X	Y	Z
0	P	0	0	1	0	0	1	0	0	0
1	Q	0	0	0	0	0	0	1	0	0
2	R	0	0	0	0	0	0	1	0	0
3	S	0	0	0	0	1	0	0	0	0
4	T	0	0	0	0	0	1	0	0	0
5	W	0	0	0	1	0	0	0	1	0
6	X	0	0	0	0	0	0	0	0	0
7	Y	0	0	1	0	0	0	0	0	1
8	Z	0	0	0	0	0	0	0	0	0

유향 그래프의 다른 예

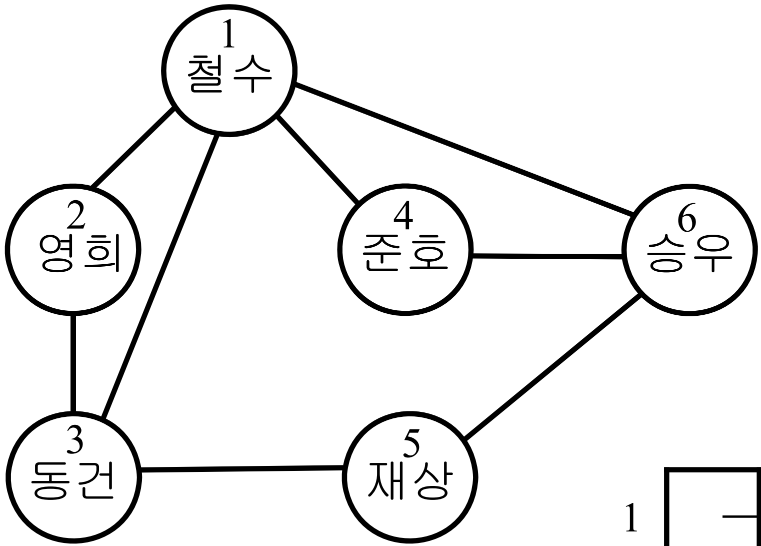


		0	1	2	3
		A	B	C	D
0	A	∞	8	∞	6
1	B	8	∞	9	∞
2	C	∞	9	∞	∞
3	D	6	∞	∞	∞

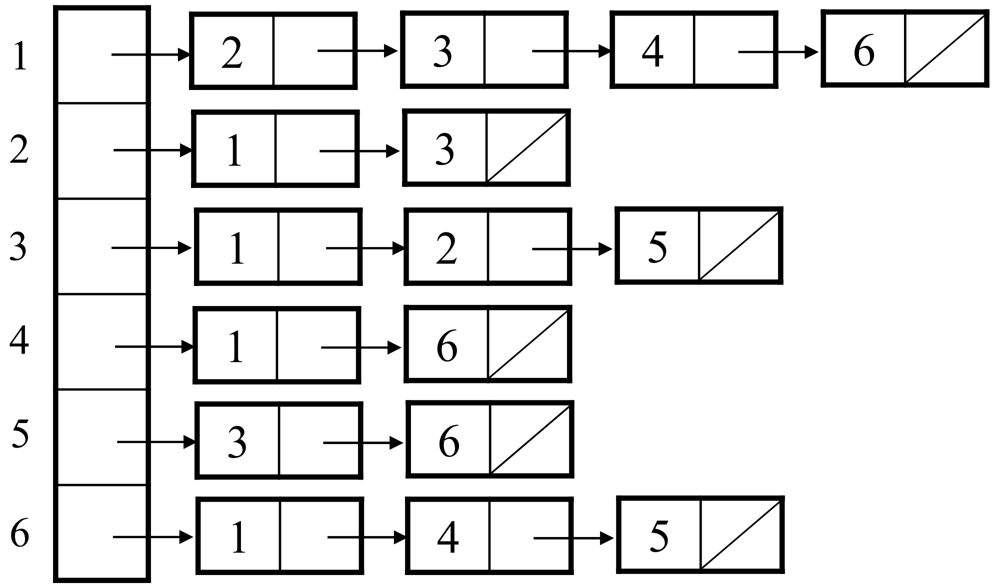
가중치 있는 그래프의 다른 예

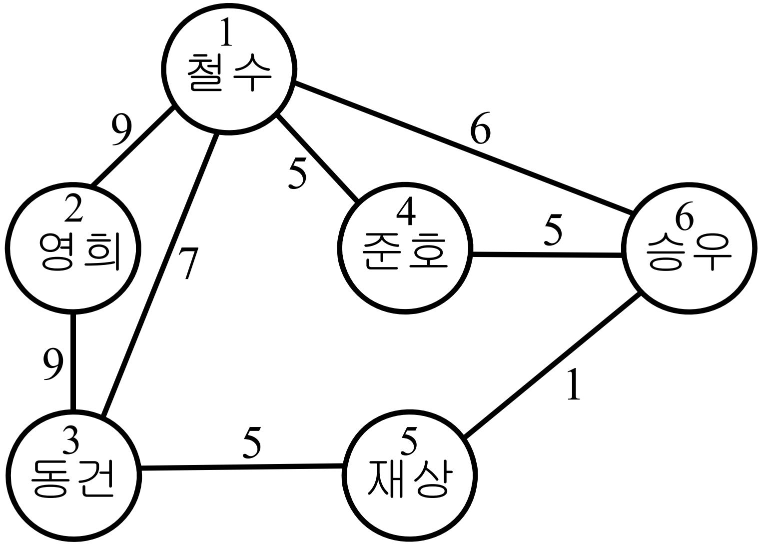
Graph의 표현 2: Adjacency List

- Adjacency list
  - N 개의 연결 리스트로 표현
  - i 번째 리스트는 정점 i 에 인접한 정점들을 리스트로 연결해 놓음
  - 가중치 있는 그래프의 경우
    - 리스트는 가중치도 보관한다

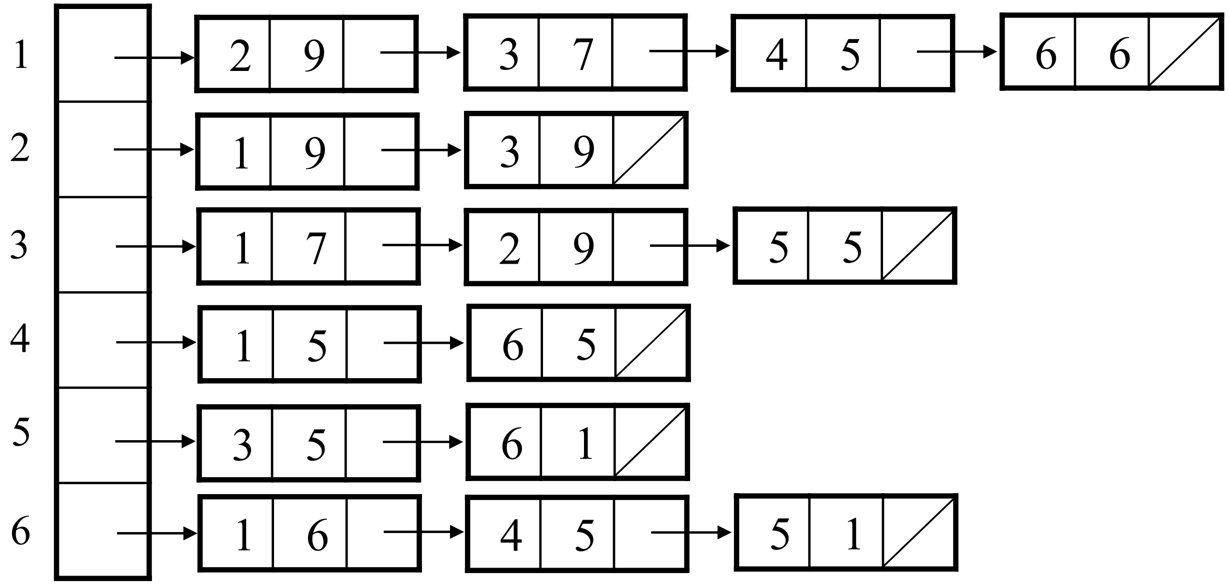


무향 그래프의 예




1 철수
2 영희
3 동건
4 준호
5 재상
6 승우
9
5
6
7
5
9
5
1

가중치 있는 그래프의 예


1
2
3
4
5
6
2 9
3 7
4 5
6 6
1 9
3 9
1 7
2 9
5 5
1 5
6 5
3 5
6 1
1 6
4 5
5 1

Graph Traversal

- 대표적 두가지 방법
  - BFS (Breadth-First Search)
  - DFS (Depth-First Search)
- 너무나 중요함
  - 그래프 알고리즘의 기본
  - DFS/BFS는 간단해 보이지만 제대로 아는 사람은 매우 드물다
  - DFS/BFS는 뼛속 깊이 이해해야 좋은 그래프 알고리즘을 만들 수 있음

DFS 깊이우선탐색

```
DFS(G)
{
        for each v ∈ V
                visited[v] ← NO;
        for each v ∈ V
                if (visited[v] = NO) then aDFS(v);
}
aDFS (v)
{
        visited[v] ← YES;
        for each x ∈ L(v)       ▷ L(v) : 정점 v의 인접 리스트
                if (visited[x] = NO) then aDFS(u);
}
```

✓수행시간: $\Theta(|V|+|E|)$

BFS너비우선탐색

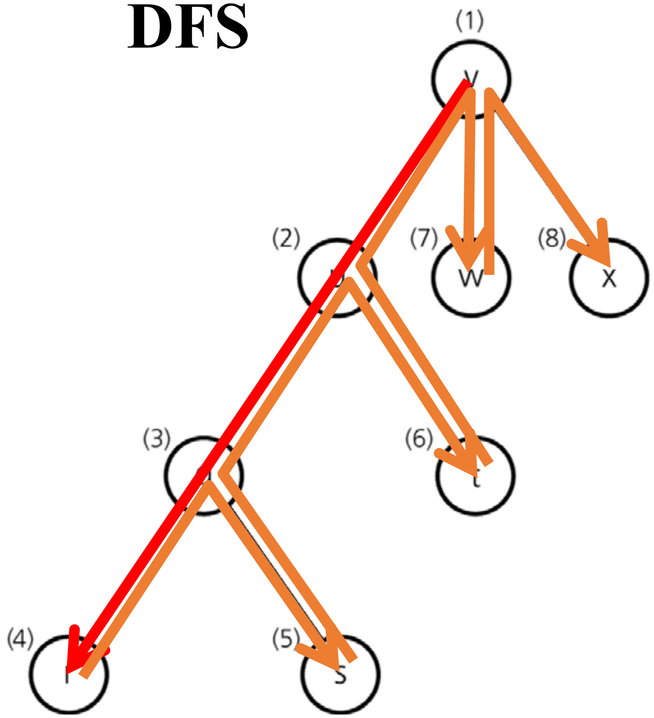
BFS(G, v)
{
  for each $v \in V$
    visited[v] ← NO;
  visited[s] ← YES;
  enqueue(Q, s);
  while ($Q \neq \Phi$) {
    u ← dequeue(Q);
    for each $v \in L(u)$
      if (visited[v] = NO) **then**
        visited[u] ← YES;
        enqueue(Q, v);
      }

  }
}

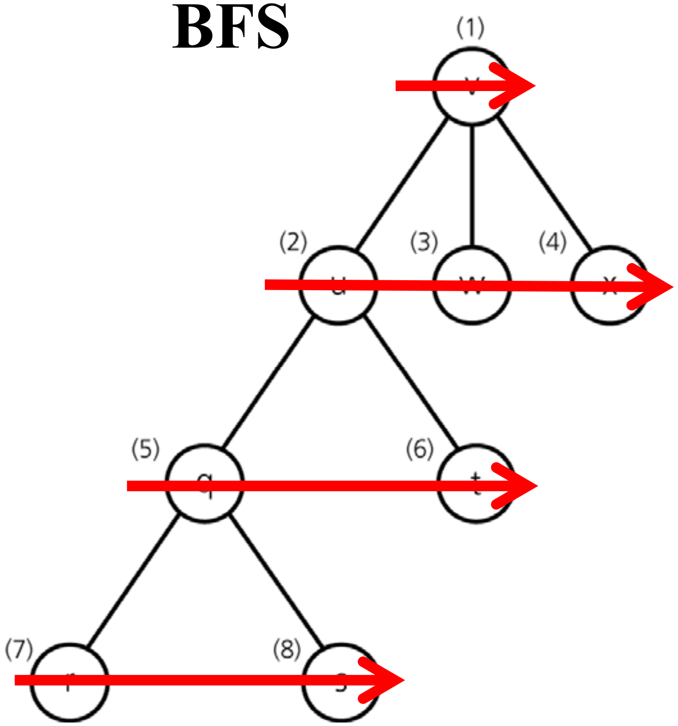
✓수행시간: $\Theta(|V|+|E|)$

동일한 Tree를 각각 DFS/BFS로 방문하기

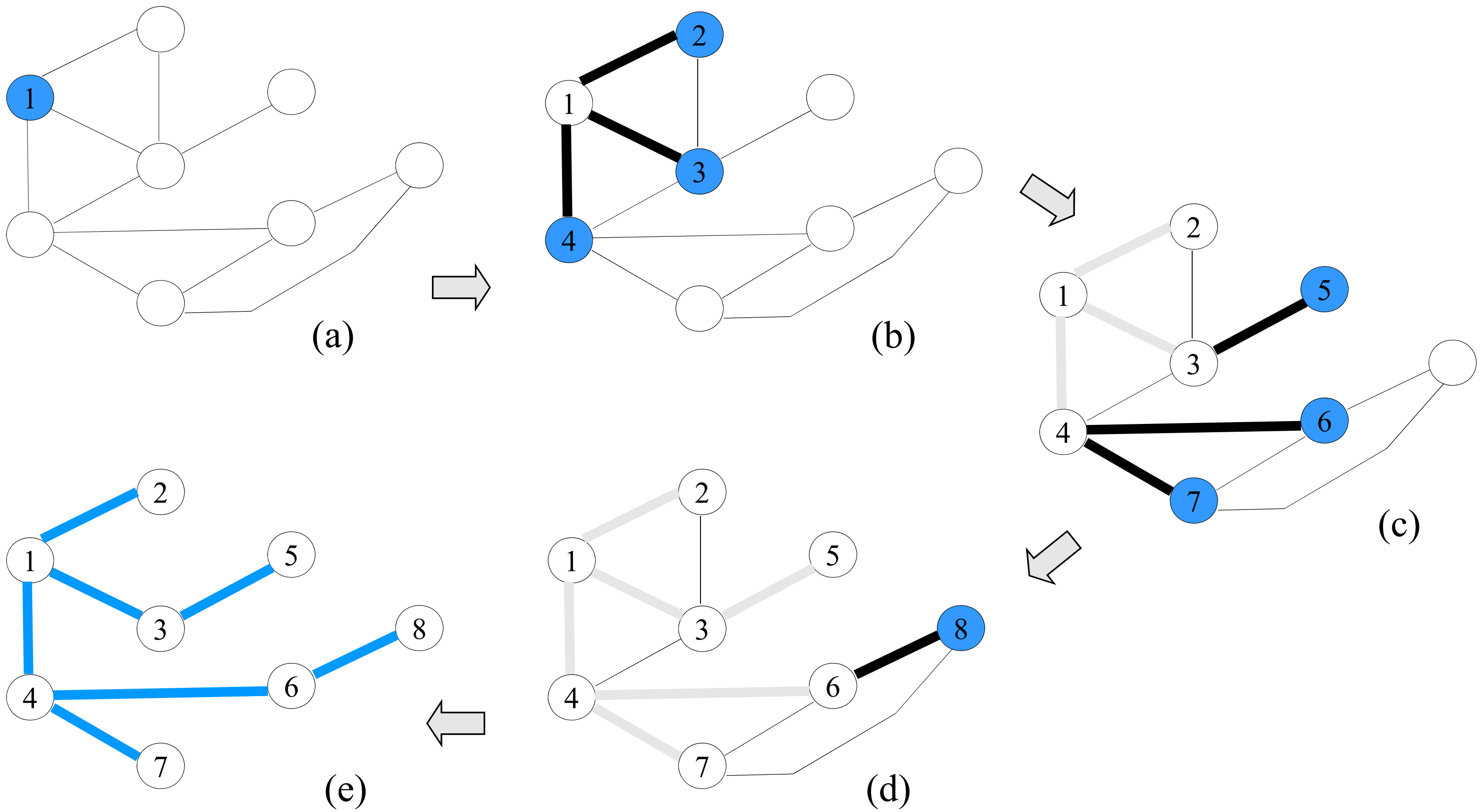
DFS



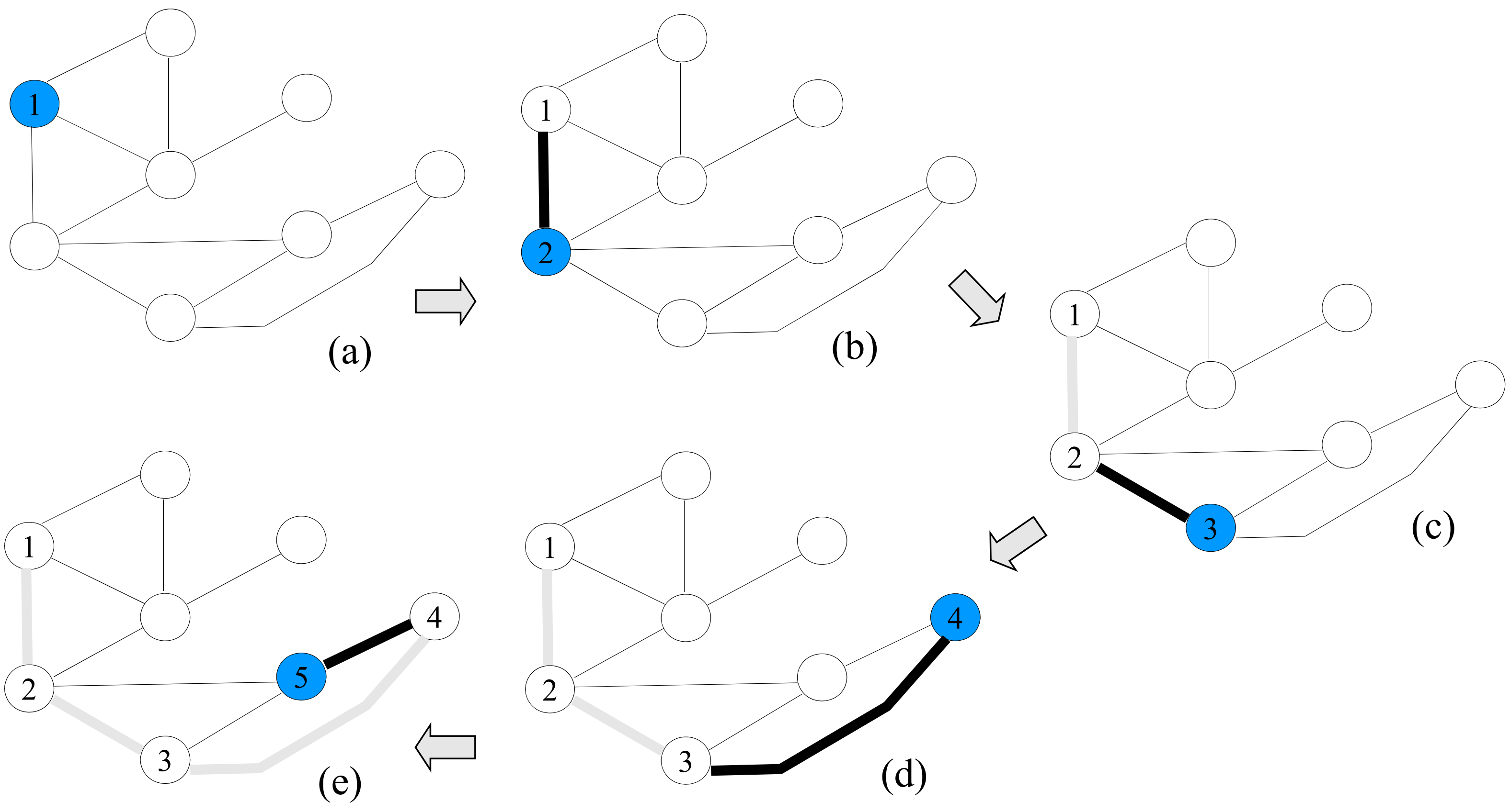
BFS



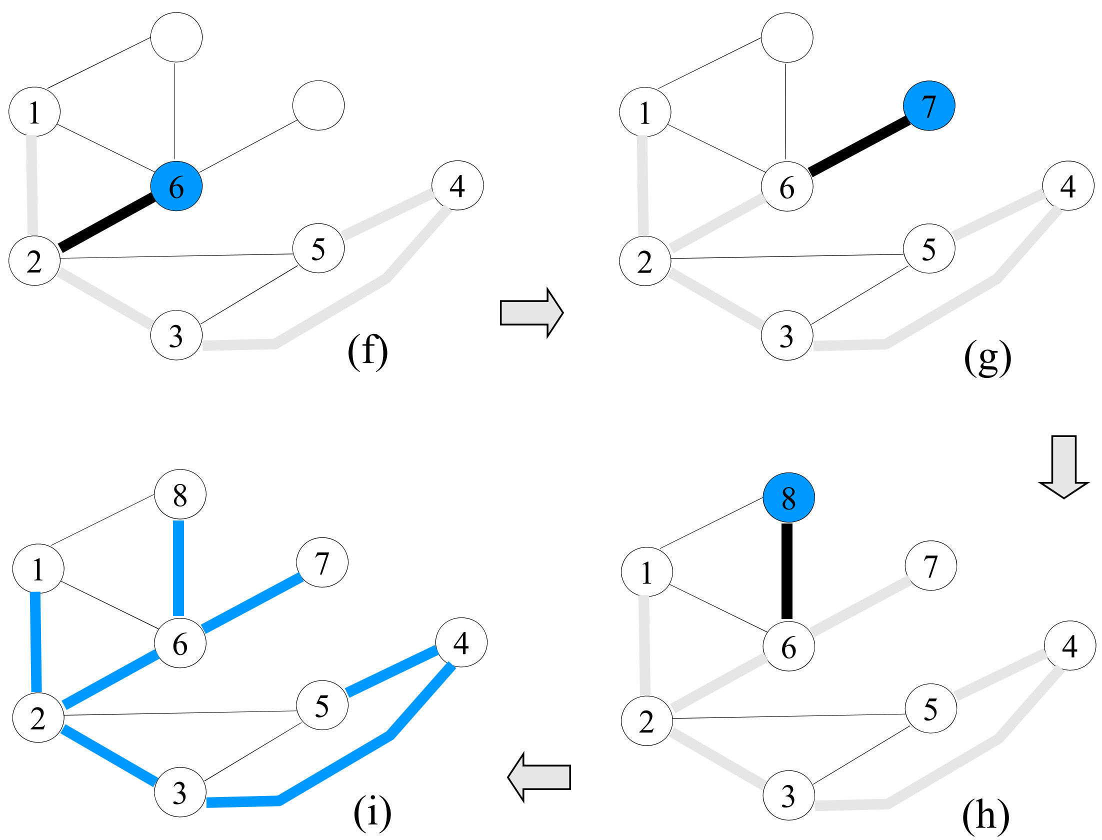
BFS의 작동 예



DFS의 작동 예



DFS의 작동 예 (계속)



Reminding – Graph Algorithms

(Prim, Kruskal, Dijkstra, Floyd-Warshall – Skip.., go to Topological Sorting)

Minimum Spanning Trees

- 조건
  - 무향 연결 그래프
    - 연결 그래프connected graph : 모든 정점 간에 경로가 존재하는 그래프
- 트리
  - 싸이클이 없는 연결 그래프
  - n 개의 정점을 가진 트리는 항상 n-1 개의 간선을 갖는다
- 그래프 G의 신장트리
  - G의 정점들과 간선들로만 구성된 트리
- G의 최소신장트리
  - G의 신장트리들 중 간선의 합이 최소인 신장트리

Prim Algorithm

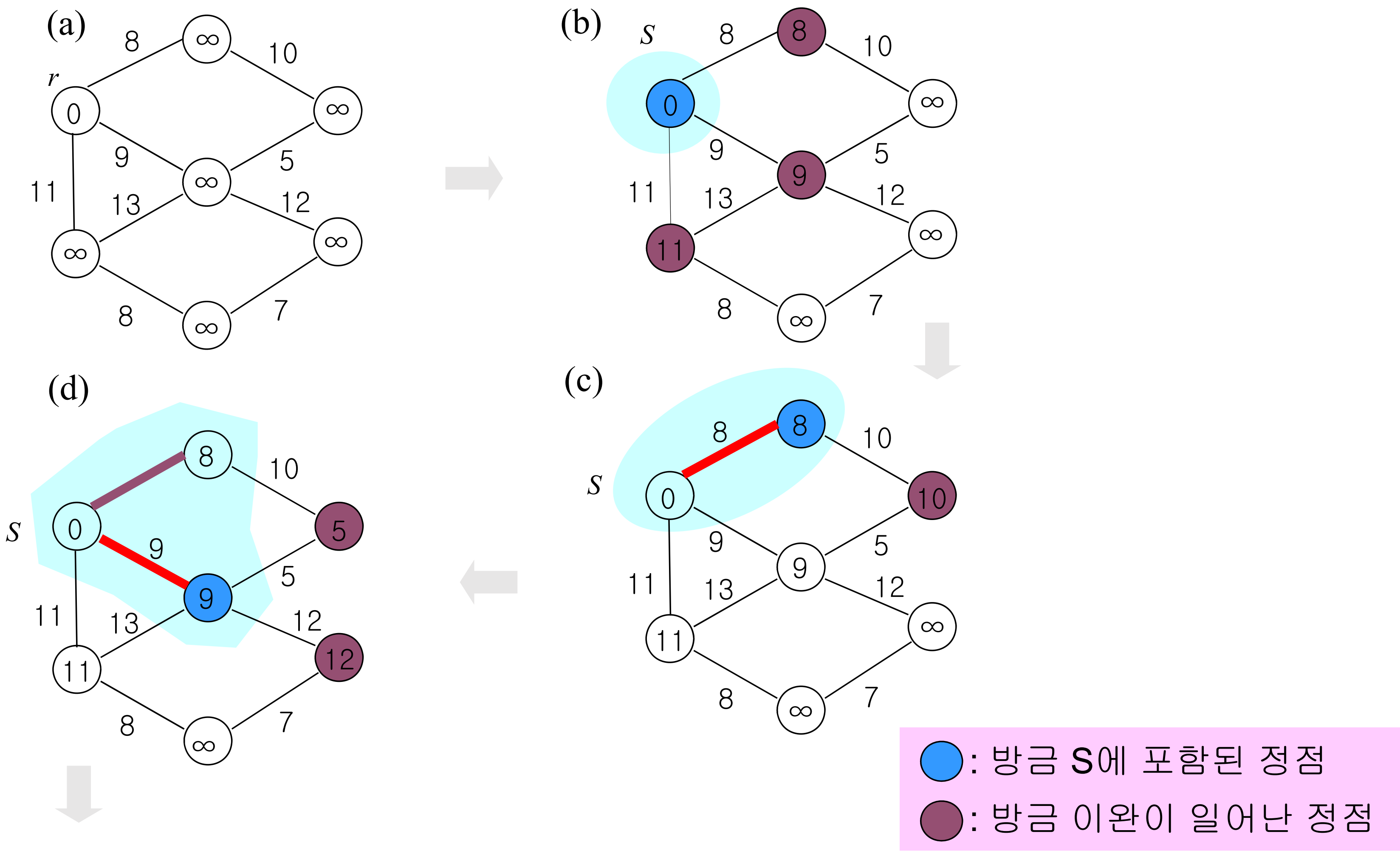
```
Prim (G, r)
{
      S ← Φ ;
      정점 r을 방문되었다고 표시하고, 집합 S에 포함시킨다;
    while (S≠V) {
              S에서 V-S를 연결하는 간선들 중 최소길이의 간선 (x,y) 를 찾는다; ▷ (x∈S, y∈V-S)
              정점 y를 방문되었다고 표시하고, 집합 S에 포함시킨다;
    }
}
```

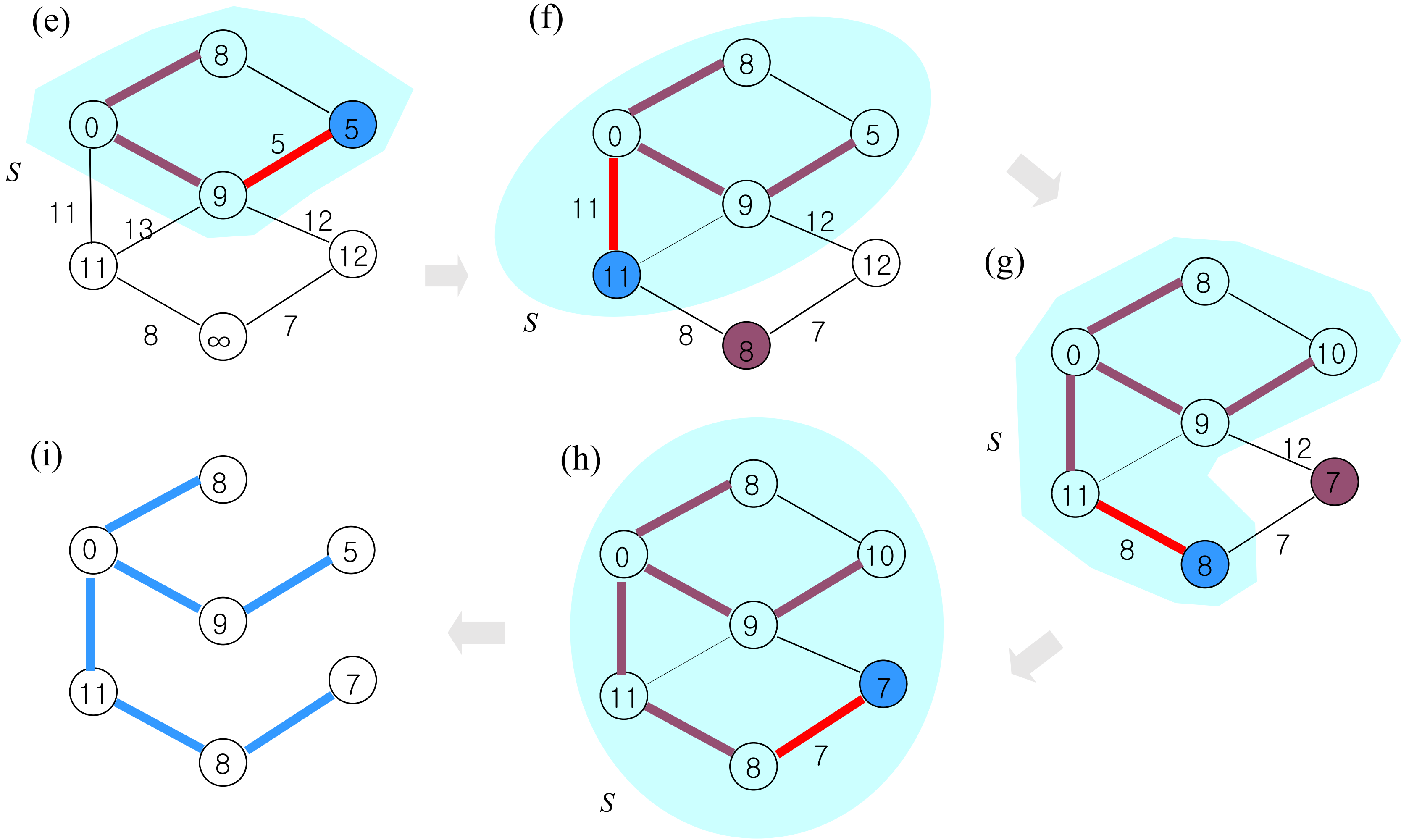
✓ Prim 알고리즘은 그리디greedy 알고리즘의 일종
✓ 그리디 알고리즘으로 최적해를 보장하는 드문 예

✓수행시간: $O(|E|\log|V|)$

힙 이용

Prim Algorithm의 작동 예




(e)
(f)
(g)
(h)
(i)
S

Kruskal Algorithm

Kruskal (G, r)

{

$T \leftarrow \Phi$; ▷ T : 신장트리

단 하나의 정점만으로 이루어진 n 개의 집합을 초기화한다;

모든 간선을 가중치가 작은 순으로 정렬한다;

while (T의 간선수 < n-1) {

최소비용 간선 (u, v)를 제거한다;

정점 u와 정점 v가 서로 다른 집합에 속하면 {

두 집합을 하나로 합친다;

$T \leftarrow T \cup \{u, v\}$;

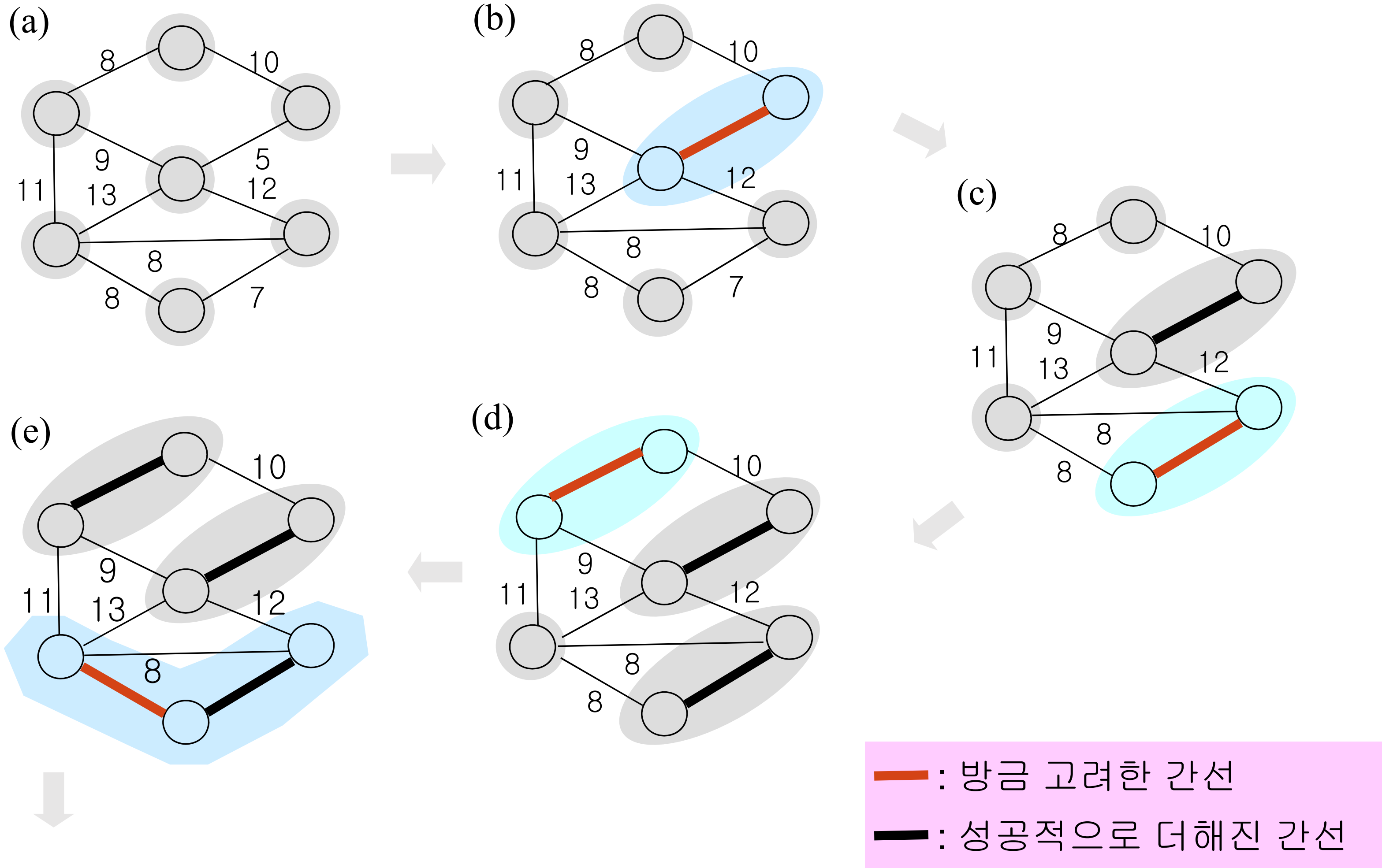
}

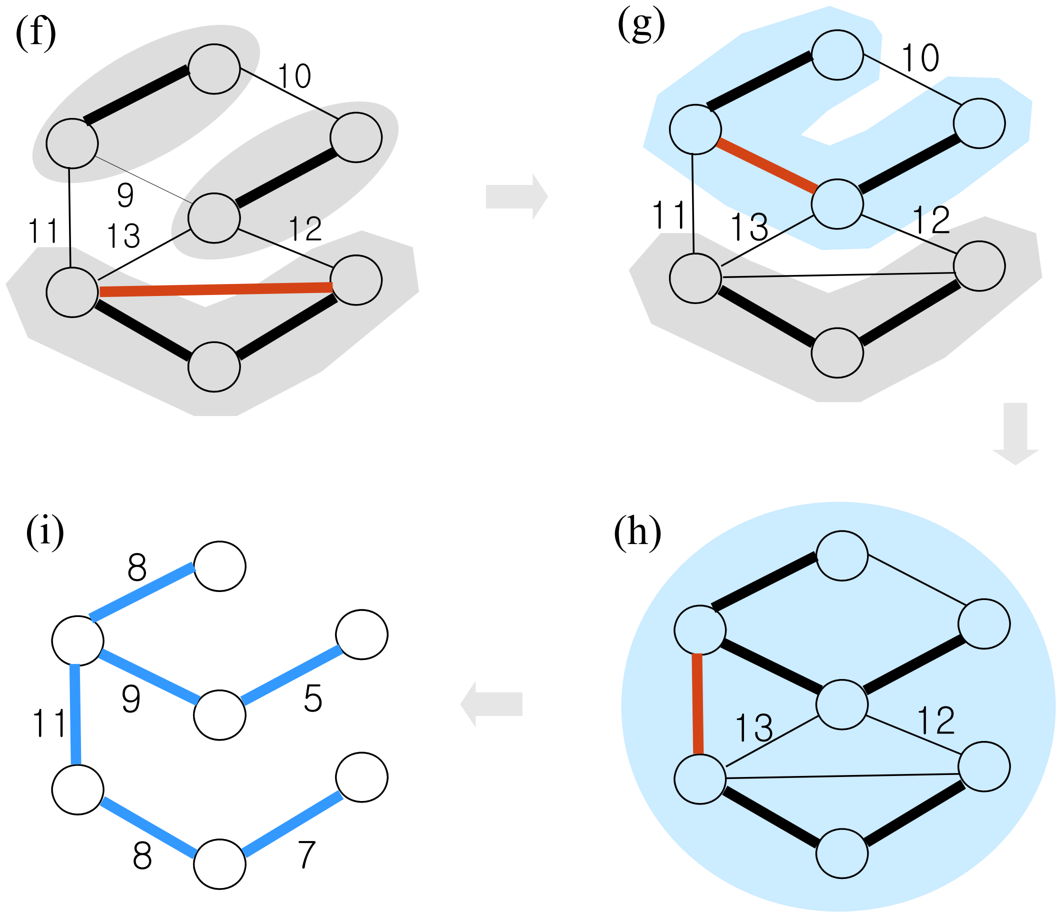
}

}

✓수행시간: $O(|E|\log|V|)$

Kruskal Algorithm의 작동 예




(f)
10
9
11
13
12
(g)
10
11
13
12
(h)
13
12
(i)
8
11
9
5
8
7

Shortest Paths

- 조건
  - 간선 가중치가 있는 유향 그래프
  - 무향 그래프는 각 간선에 대해 양쪽으로 유향 간선이 있는 유향 그래프로 생각할 수 있다
    - 즉, 무향 간선 (u,v)는 유향 간선 (u,v)와 (v,u)를 의미한다고 가정하면 된다
- 두 정점 사이의 최단경로
  - 두 정점 사이의 경로들 중 간선의 가중치 합이 최소인 경로
  - 간선 가중치의 합이 음인 싸이클이 있으면 문제가 정의되지 않는다

- 단일 시작점 최단경로
  - 단일 시작점으로부터 각 정점에 이르는 최단경로를 구한다
  - ➢ 다익스트라 알고리즘
    - 음의 가중치를 허용하지 않는 최단경로
  - ➢ 벨만-포드 알고리즘
    - 음의 가중치를 허용하는 최단경로
  - ➢ 싸이클이 없는 그래프의 최단경로
- 모든 쌍 최단경로
  - 모든 정점 쌍 사이의 최단경로를 모두 구한다
  - ➢ 플로이드-워샬 알고리즘

Dijkstra Algorithm

모든 간선의 가중치는 음이 아니어야 함

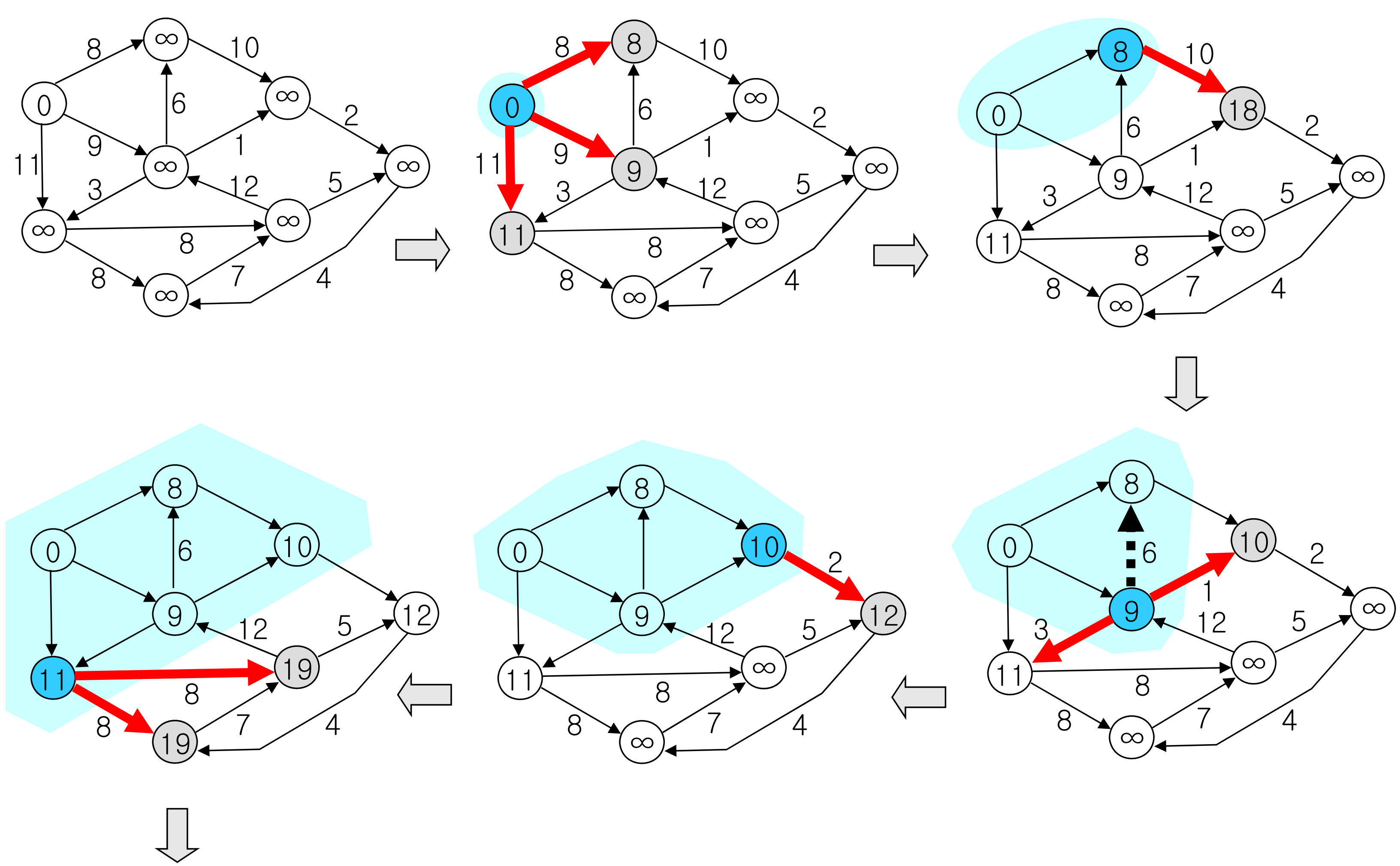
```
Dijkstra(G, r)
▷ G=(V, E): 주어진 그래프
▷ r: 시작으로 삼을 정점
{
    S ← Φ ;                ▷ S : 정점 집합
    for each u∈V
        d_u ← ∞ ;
    d_r ← 0 ;
    while (S≠V){           ▷ n회 순환된다
        u ← extractMin(V-S, d) ;
        S ← S ∪ {u};
        for each v∈L(u) ▷ L(u) : u로부터 연결된 정점들의 집합
            if (v∈V-S and d_v<d_u+w_{u,v}) then d_v ← d_u+w_{u,v} ;
    }
}

extractMin(Q, d)
{
    집합 Q에서 d값이 가장 작은 정점 u를 리턴한다 ;
}
```

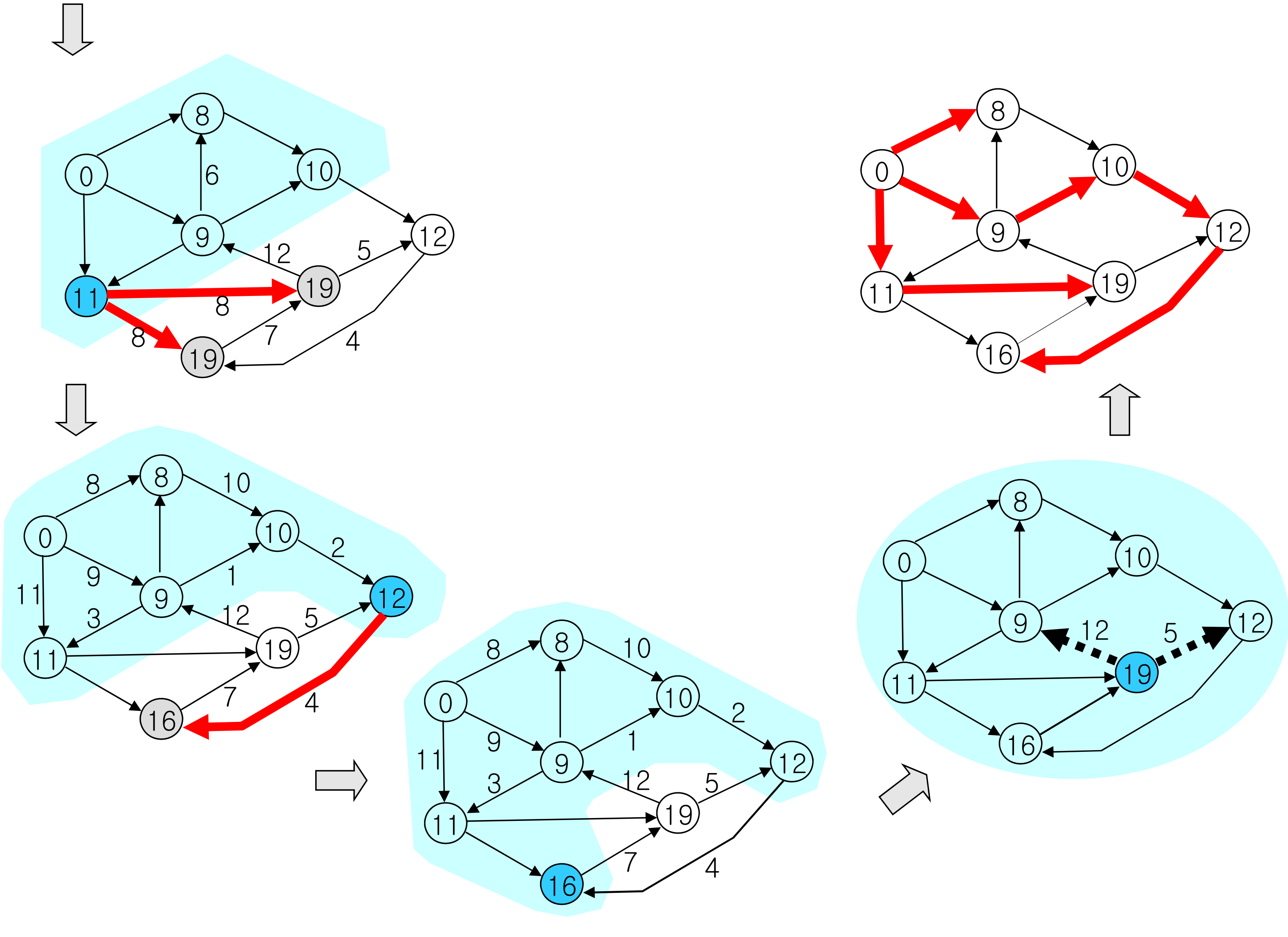
이완(relaxation)

✓수행시간: $O(|E|\log|V|)$

힙 이용

Dijkstra Algorithm의 작동 예




0 8 9 10 11 12 19 19
6 12 5 8 8 7 4
0 8 9 10 11 12 16 19
8 10 2 11 9 1 3 12 5 7 4
0 8 9 10 11 12 16 19
8 10 2 11 9 1 3 12 5 7 4
0 8 9 10 11 12 16 19
12 5
0 8 9 10 11 12 16 19

Floyd-Warshall Algorithm

- 모든 정점들간의 상호 최단거리 구하기
- 응용 예
  - Road Atlas
  - 네비게이션 시스템
  - 네트웍 커뮤니케이션

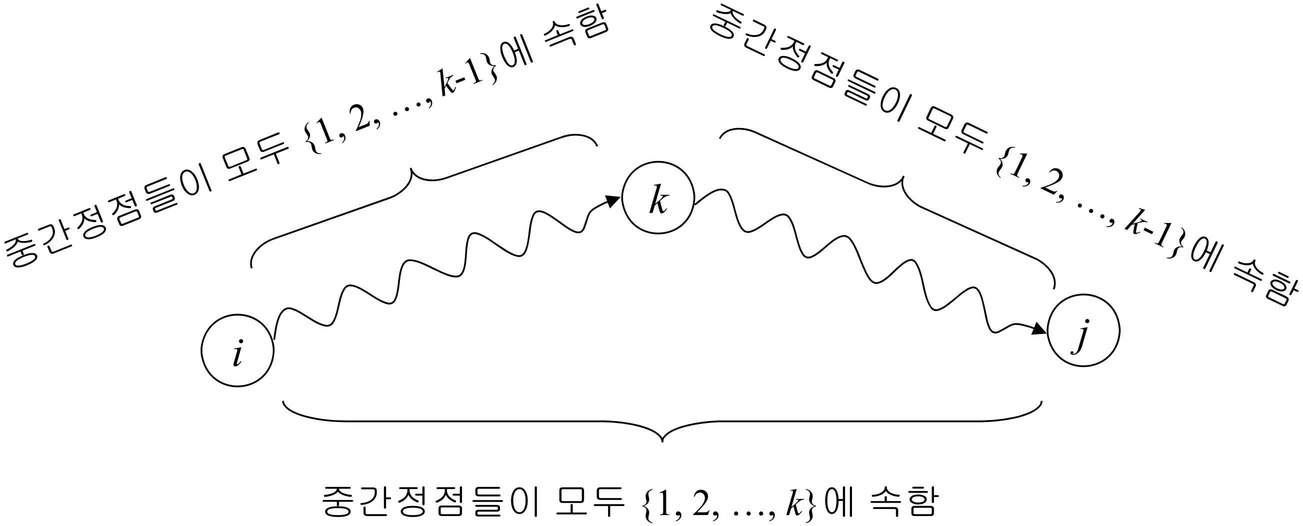
Floyd-Warshall Algorithm

FloydWarshall(G)
{
    for $i \leftarrow 1$ **to** n
        for $j \leftarrow 1$ **to** n
            $d^0_{ij} \leftarrow w_{ij}$;
    for $k \leftarrow 1$ **to** n ▷ 중간정점 집합 $\{1, 2, \ldots, k\}$
        for $i \leftarrow 1$ **to** n ▷ i : 시작 정점
            for $j \leftarrow 1$ **to** n ▷ j : 마지막 정점
                $d^k_{ij} \leftarrow \min \{d^{k-1}_{ij}, d^{k-1}_{ik} + d^{k-1}_{kj}\}$;
}

✓ d^k_{ij} : 중간 정점으로 정점 집합 $\{1, 2, \ldots, k\}$만을 사용하여 정점 i에서 정점 j에 이르는 최단경로

✓수행시간: $\Theta(|V|^3)$

d^{k}_{ij} 관련



싸이클이 없는 Graph의 Shortest Path

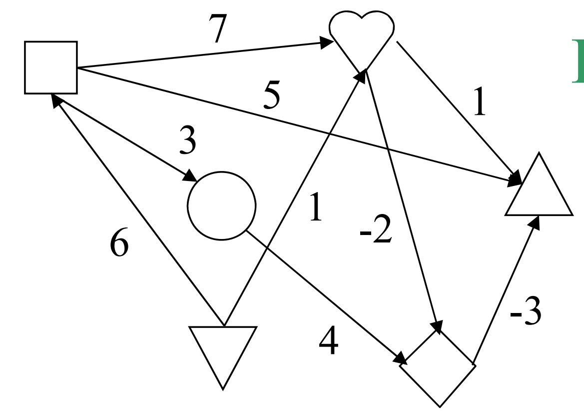
- 싸이클이 없는 유향 그래프를 DAG라 한다
  - DAG: Directed Acyclic Graph
- DAG에서의 최단경로는 선형시간에 간단히 구할 수 있다

```
DAG-ShortestPath(G, r)
{
    for each u∈V
        d_u ← ∞;
    d_r ← 0;
    G의 정점들을 위상정렬한다;
    for each u∈V (위상정렬 순서로)
        for each v∈L(u)  ▷ L(u) : 정점 u로부터 연결된 정점들의 집합
            if (d_u + w_{u,v} < d_v) then d_v ← d_u + w_{u,v} ;
}
```

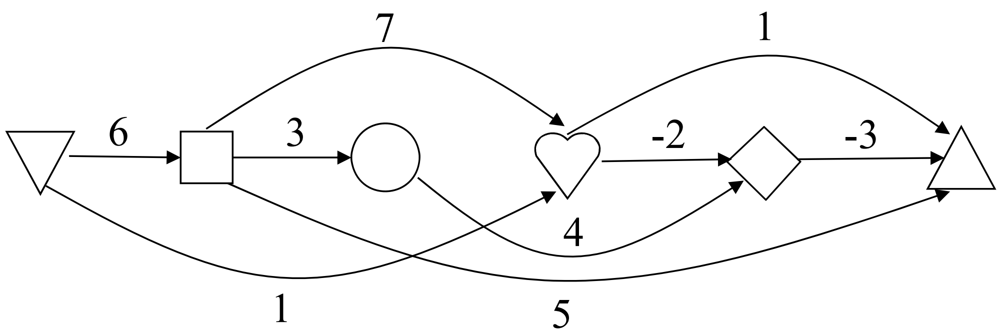
✓수행시간: $\Theta(|V|+|E|)$

DAG-ShortestPath의 작동 예

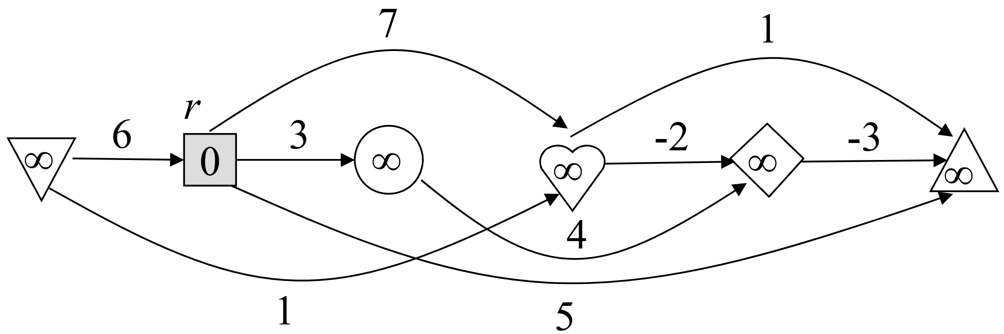
(a)

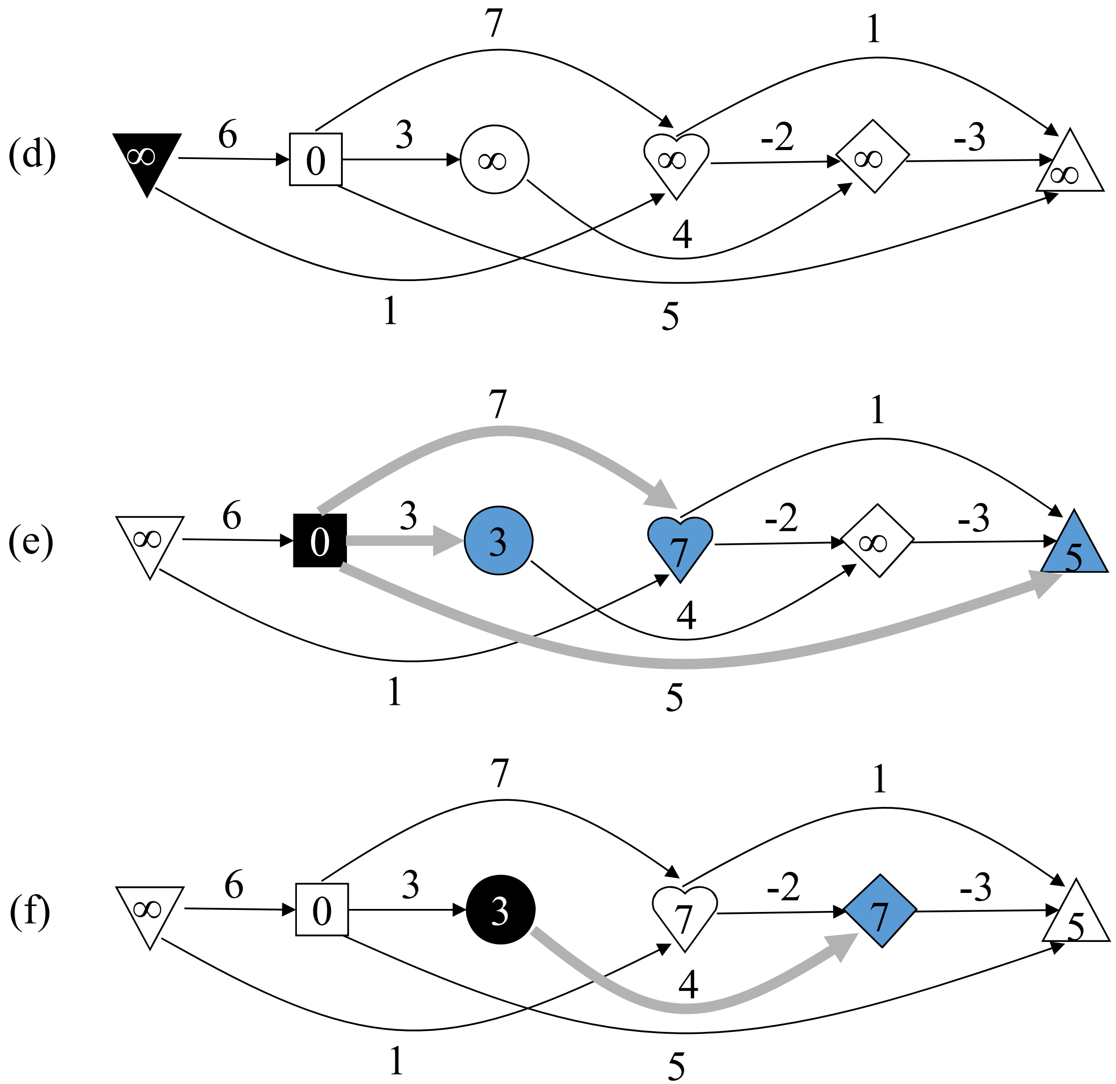


(b)

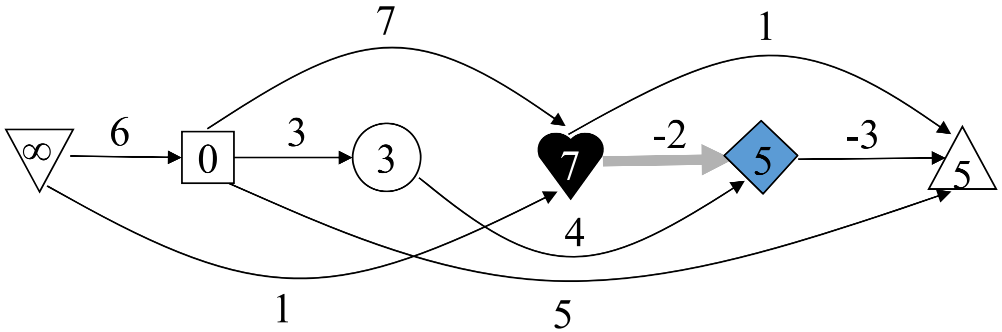


(c)

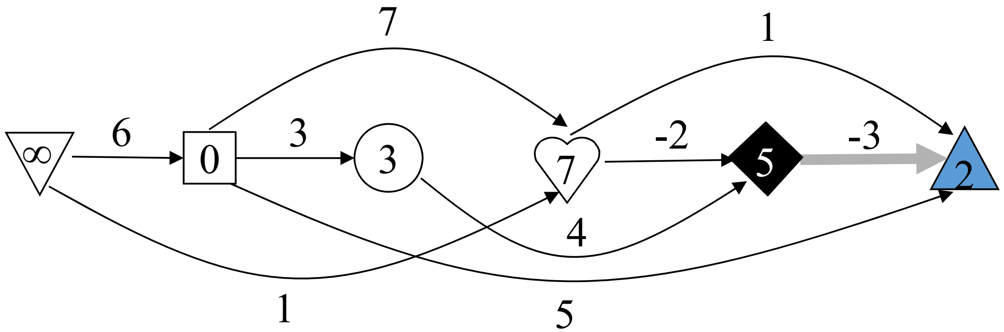



(d)
6
3
7
1
-2
-3
4
1
5
0
∞
(e)
6
3
7
1
-2
-3
4
1
5
0
3
7
5
∞
(f)
6
3
7
1
-2
-3
4
1
5
0
3
7
7
5
∞

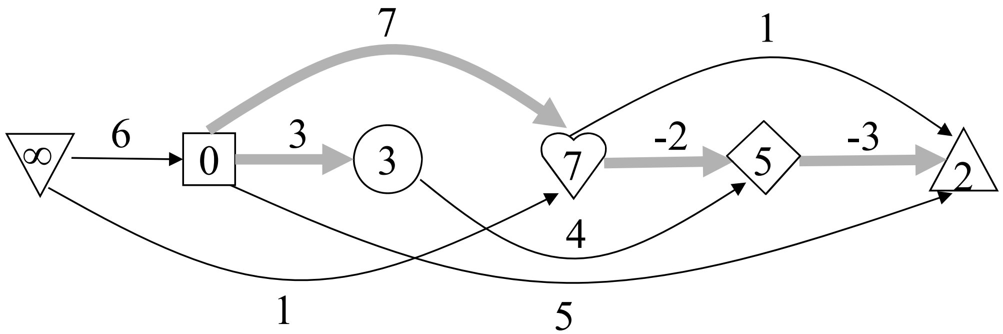
(g)



(h)



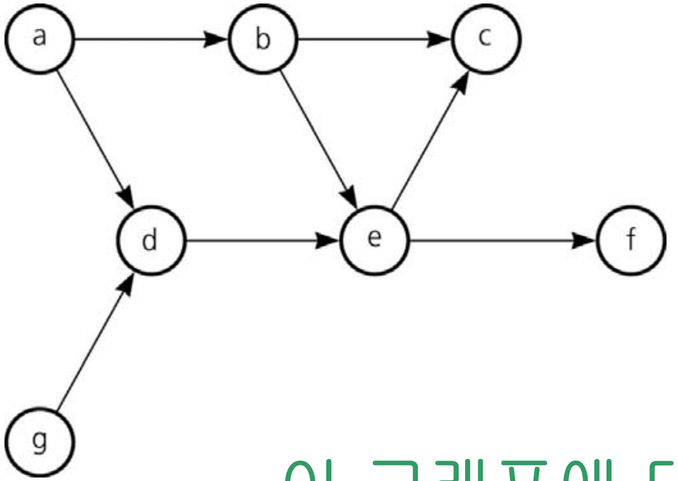
(i)



Topological Sorting

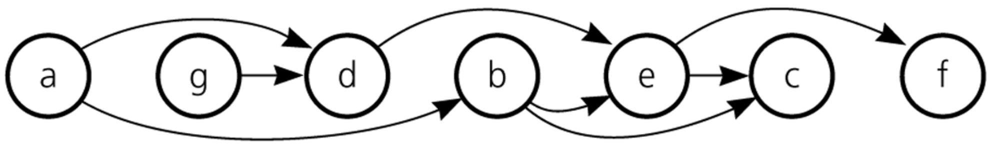
Topological Sorting

- 조건
  - 싸이클이 없는 유향 그래프
- Topological Sorting위상정렬
  - 모든 정점을 일렬로 나열하되
  - 정점 x에서 정점 y로 가는 간선이 있으면 x는 반드시 y보다 앞에 위치한다
  - 일반적으로 임의의 유향 그래프에 대해 복수의 위상 순서가 존재한다

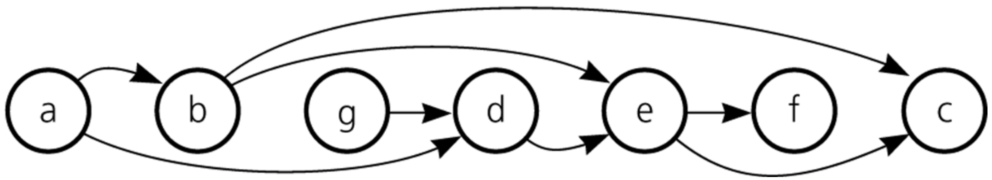


이 그래프에 대한 위상정렬의 예 2개

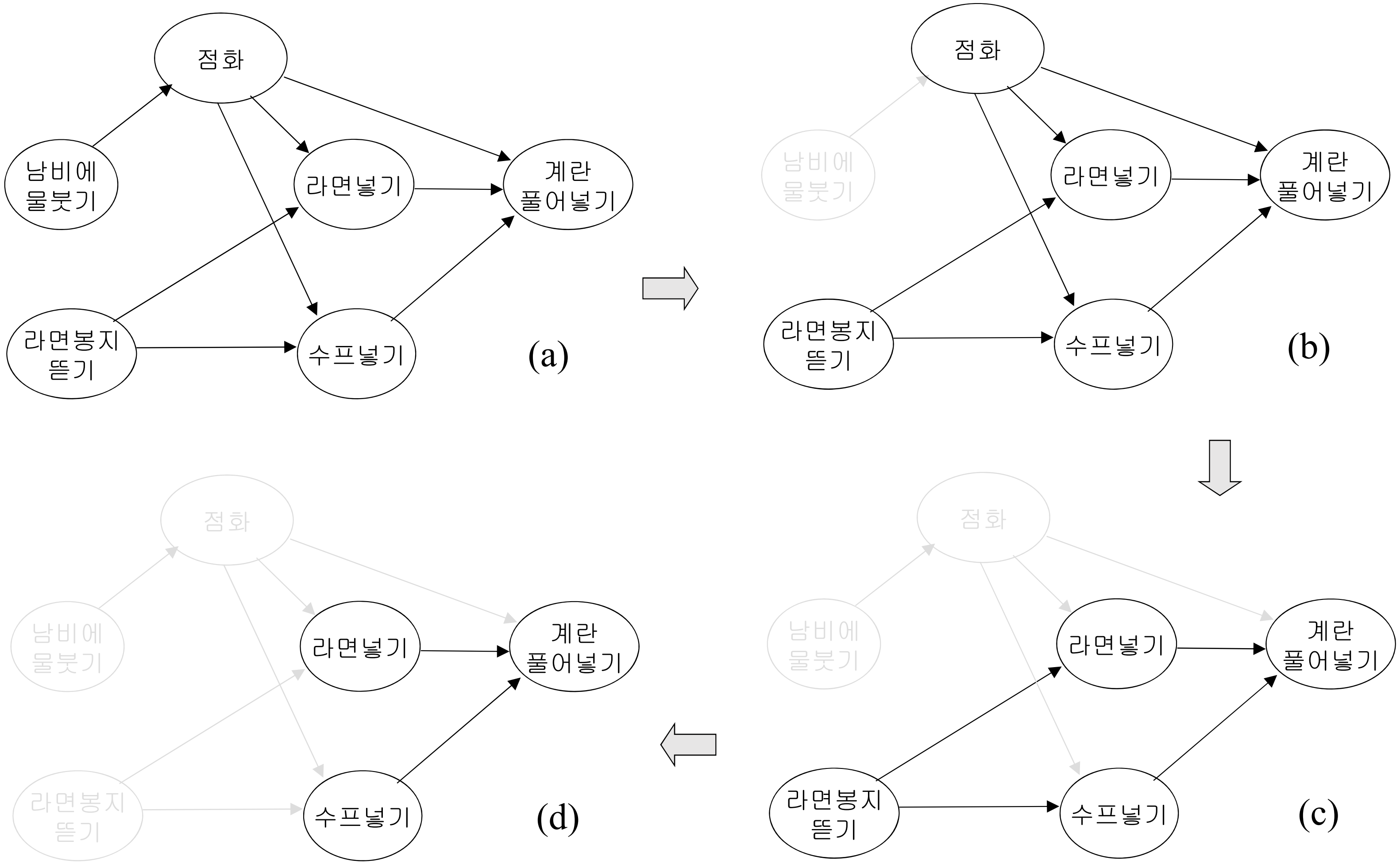
(a)

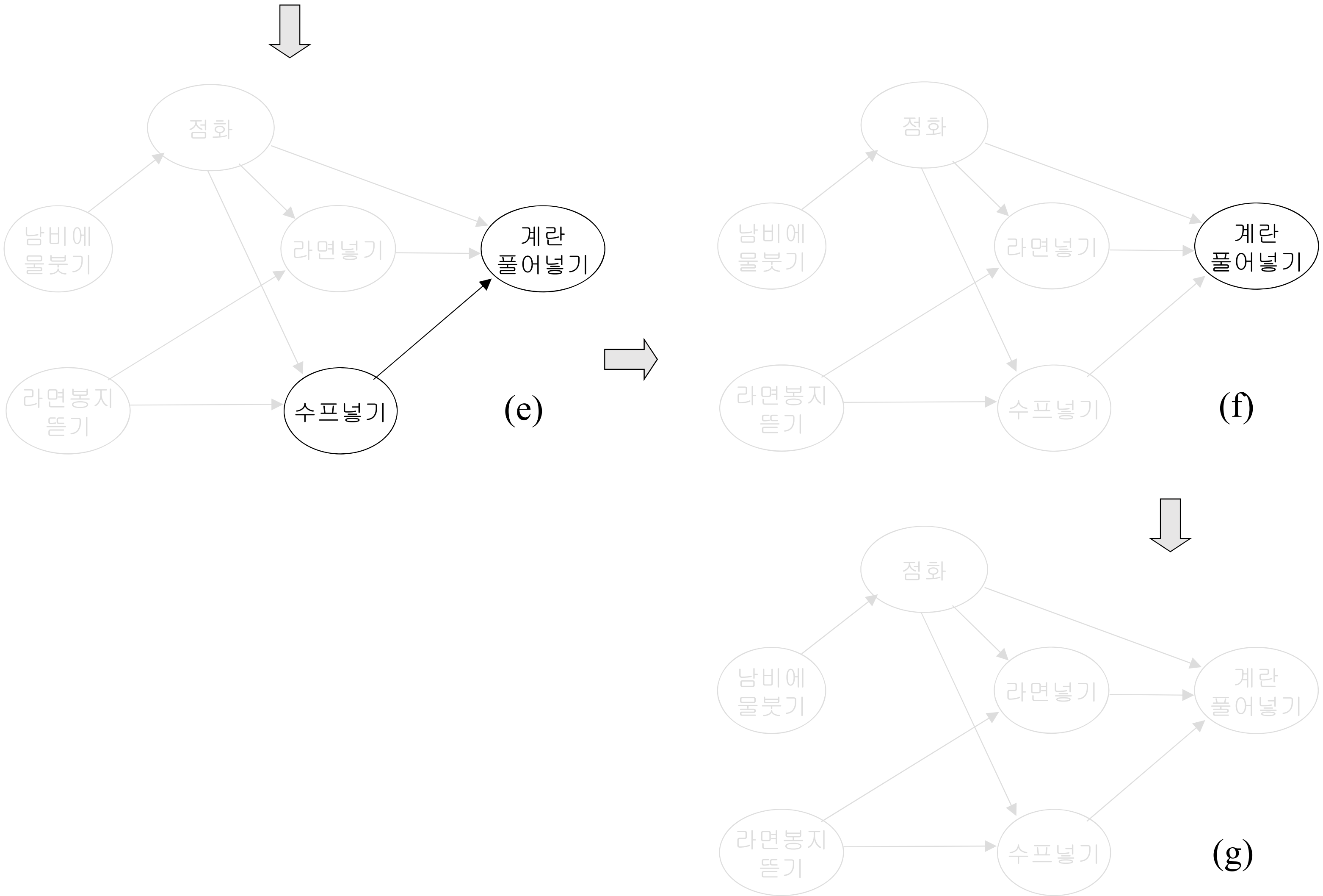


(b)



위상정렬 알고리즘 1의 작동 예





점화
남비에 물붓기
라면넣기
계란 풀어넣기
라면봉지 뜯기
수프넣기
(e)
(f)
(g)

위상정렬 알고리즘 1
(Khan's Algorithm)

topologicalSort1(G)
{
    for ← 1 **to** n {
        진입간선이 없는 정점 u를 선택한다;   // How to implement?
        $A[i] \leftarrow u$;
        정점 u와, u의 진출간선을 모두 제거한다;
    }
    ▷ 이 시점에 배열 $A[1…n]$에는 정점들을 위상정렬 되어 있다
}

✓수행시간: $\Theta(|V|+|E|)$

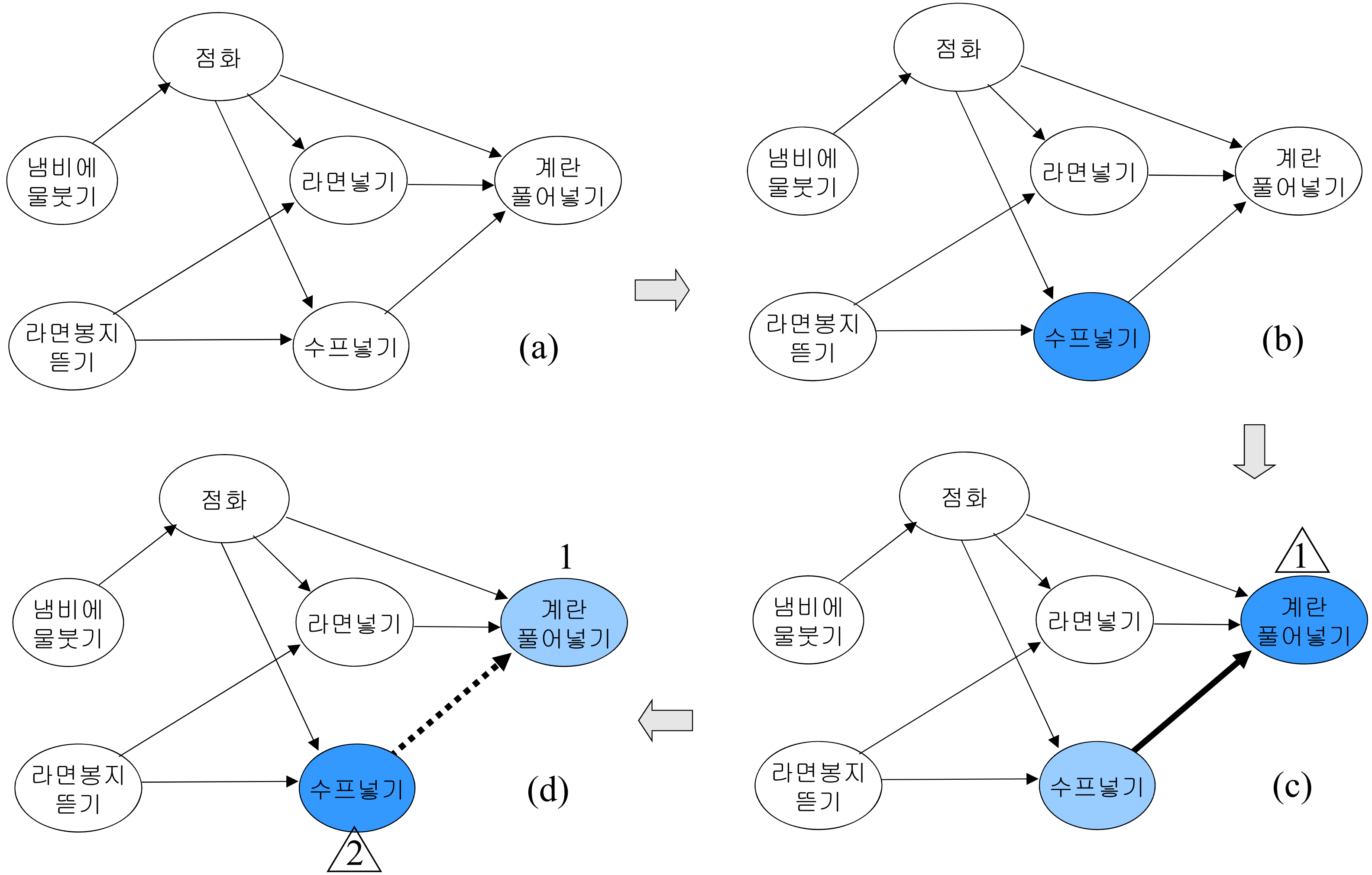
위상정렬 알고리즘 1

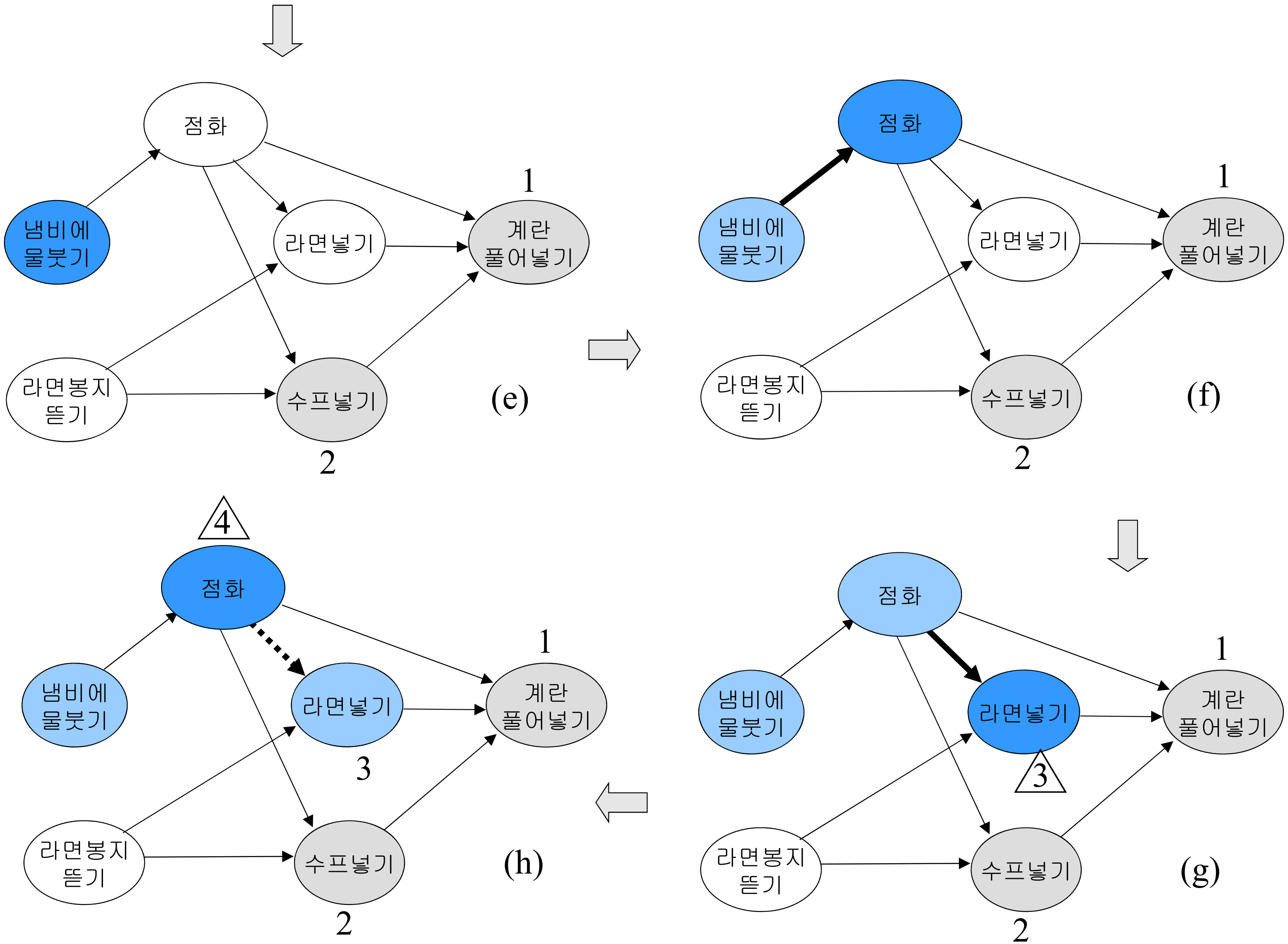
- preprocessing: build a set I of the nodes without in-edges
- Counts the number of in-edges

```
topologicalSort1(G)
{
    for ← 1 to n {
        진입간선이 없는 정점 u를 선택한다;     // How to implement?
        A[i] ← u;
        정점 u와, u의 진출간선을 모두 제거한다;
        // when the edges are deleted, decrease the counts of in-edges of the linked nodes
        // if the number of in-edges of the linked nodes == 0, then add the nodes to I
    }
    ▷ 이 시점에 배열 A[1…n]에는 정점들을 위상정렬 되어 있다
}
```

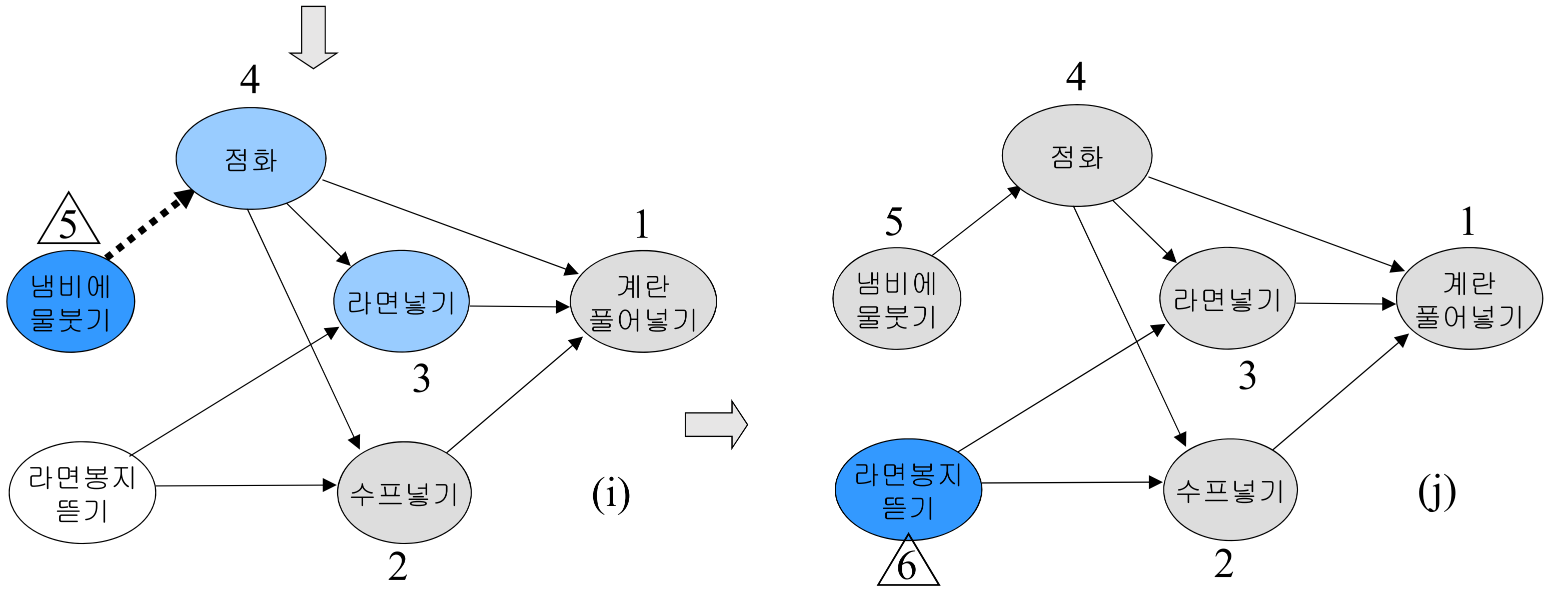
✓수행시간: $\Theta(|V|+|E|)$

위상정렬 알고리즘 2의 작동 예





점화
냄비에
물붓기
라면넣기
계란
풀어넣기
라면봉지
뜯기
수프넣기
1
2
(e)
점화
냄비에
물붓기
라면넣기
계란
풀어넣기
라면봉지
뜯기
수프넣기
1
2
(f)
4
점화
냄비에
물붓기
라면넣기
계란
풀어넣기
라면봉지
뜯기
수프넣기
1
2
3
(h)
점화
냄비에
물붓기
라면넣기
계란
풀어넣기
라면봉지
뜯기
수프넣기
1
2
3
(g)


4
점화
5
냄비에
물붓기
3
라면넣기
1
계란
풀어넣기
라면봉지
뜯기
수프넣기
2
(i)
4
점화
5
냄비에
물붓기
1
계란
풀어넣기
라면넣기
3
라면봉지
뜯기
6
수프넣기
2
(j)

위상정렬 알고리즘 2

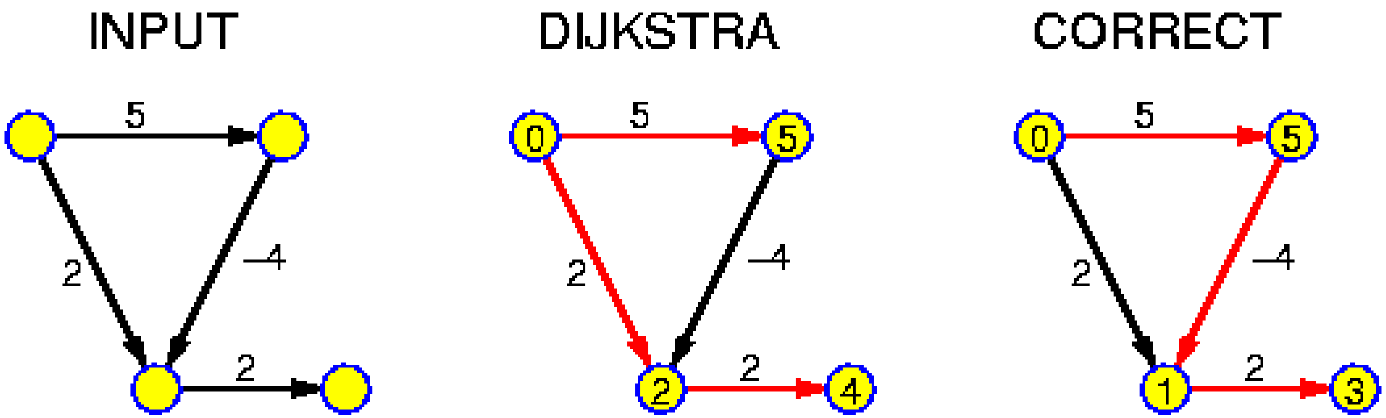
```
topologicalSort2(G)
{
    for each v∈V
        visited[v] ← NO;
    for each v∈V    ▷ 정점의 순서는 아무 순서나 무관
        if (visited[v] = NO) then DFS-TS(v) ;
}
DFS-TS(v)
{
    visited[v] ← YES;
    for each x∈L(v)    ▷ L(v): v의 인접 리스트
        if (visited[x] = NO) then DFS-TS(x) ;
    연결 리스트 R의 맨 앞에 정점 v를 삽입한다;
}
```

✓수행시간: $\Theta(|V|+|E|)$

✓알고리즘이 끝나고 나면 연결 리스트 R에는 정점들이 위상정렬된 순서로 매달려 있다.

Bellman-Ford Algorithm

NEGATIVE EDGE WEIGHTS



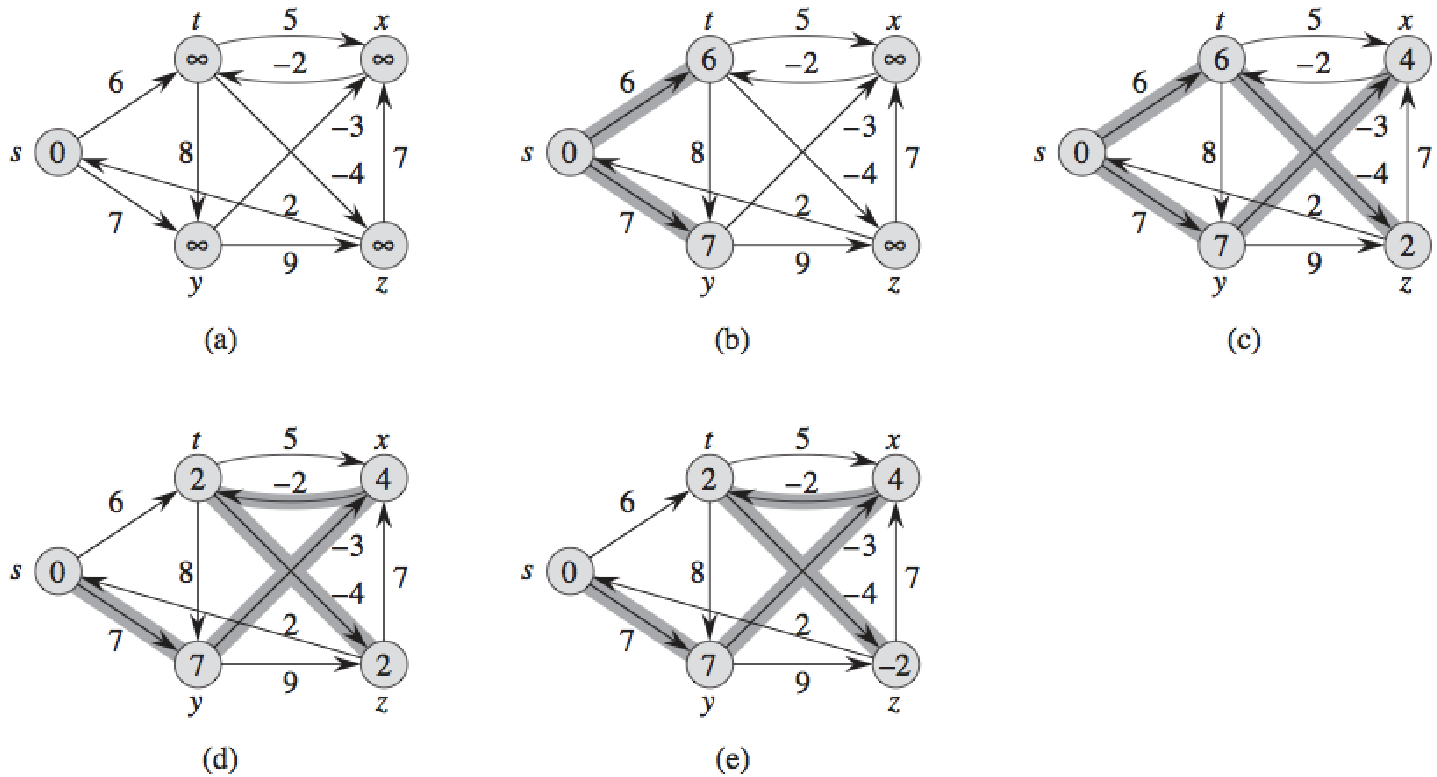


Figure 24.4 The execution of the Bellman-Ford algorithm. The source is vertex s. The d values appear within the vertices, and shaded edges indicate predecessor values: if edge (u, v) is shaded, then $v.\pi = u$. In this particular example, each pass relaxes the edges in the order $(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$. **(a)** The situation just before the first pass over the edges. **(b)–(e)** The situation after each successive pass over the edges. The d and π values in part (e) are the final values. The Bellman-Ford algorithm returns TRUE in this example.

Bellman-Ford Algorithm

음의 가중치를 허용한다

BellmanFord(G, r)

{

for each $u \in V$

$d_u \leftarrow \infty$;

$d_r \leftarrow 0$;

for $i \leftarrow 1$ **to** $|V|$-1

for each $(u, v) \in E$

if $(d_u + w_{u,v} < d_v)$ **then** $d_v \leftarrow d_u + w_{u,v}$;

이완(relaxation)

}

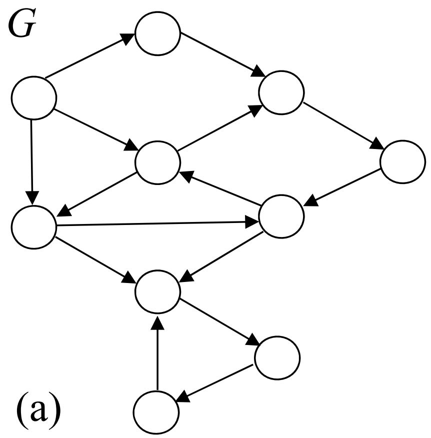
✓수행시간: $\Theta(|E||V|)$

Strongly Connected Component Detection

Strongly Connected Components

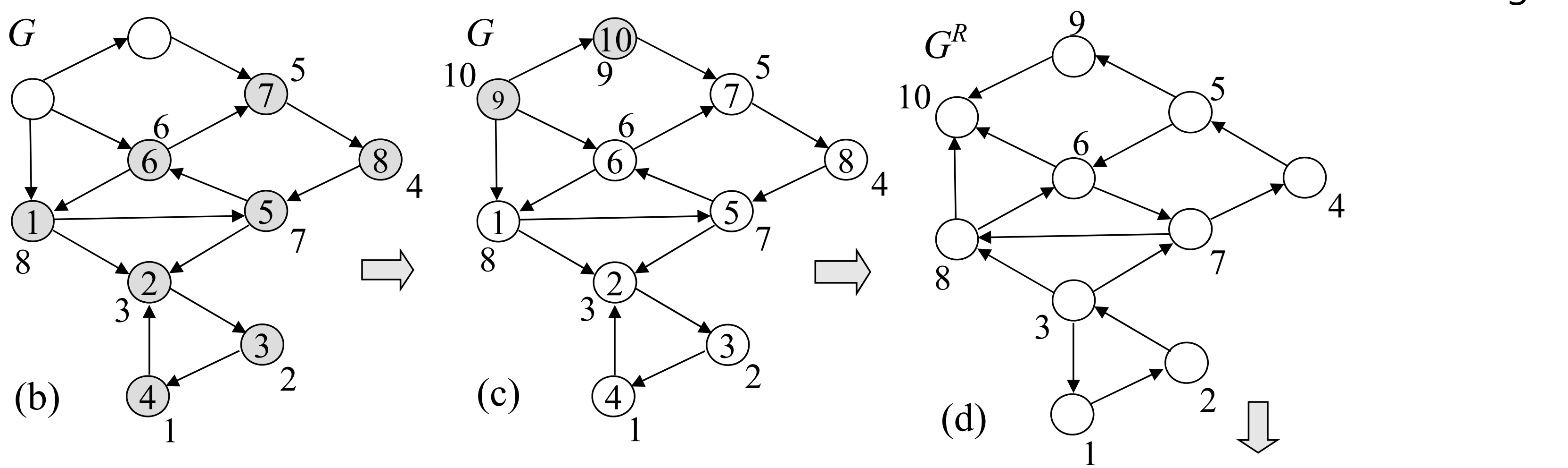
- in directed graphs
- “Strongly connected”
  - 그래프의 모든 정점쌍에 대해서 양방향으로 경로가 존재하면 강하게 연결되었다고 한다
  - 포함된 모든 요소가 강하게 연결된 부분 그래프 -> strongly connected component
- How to find the strongly connected component from a given directed graph?

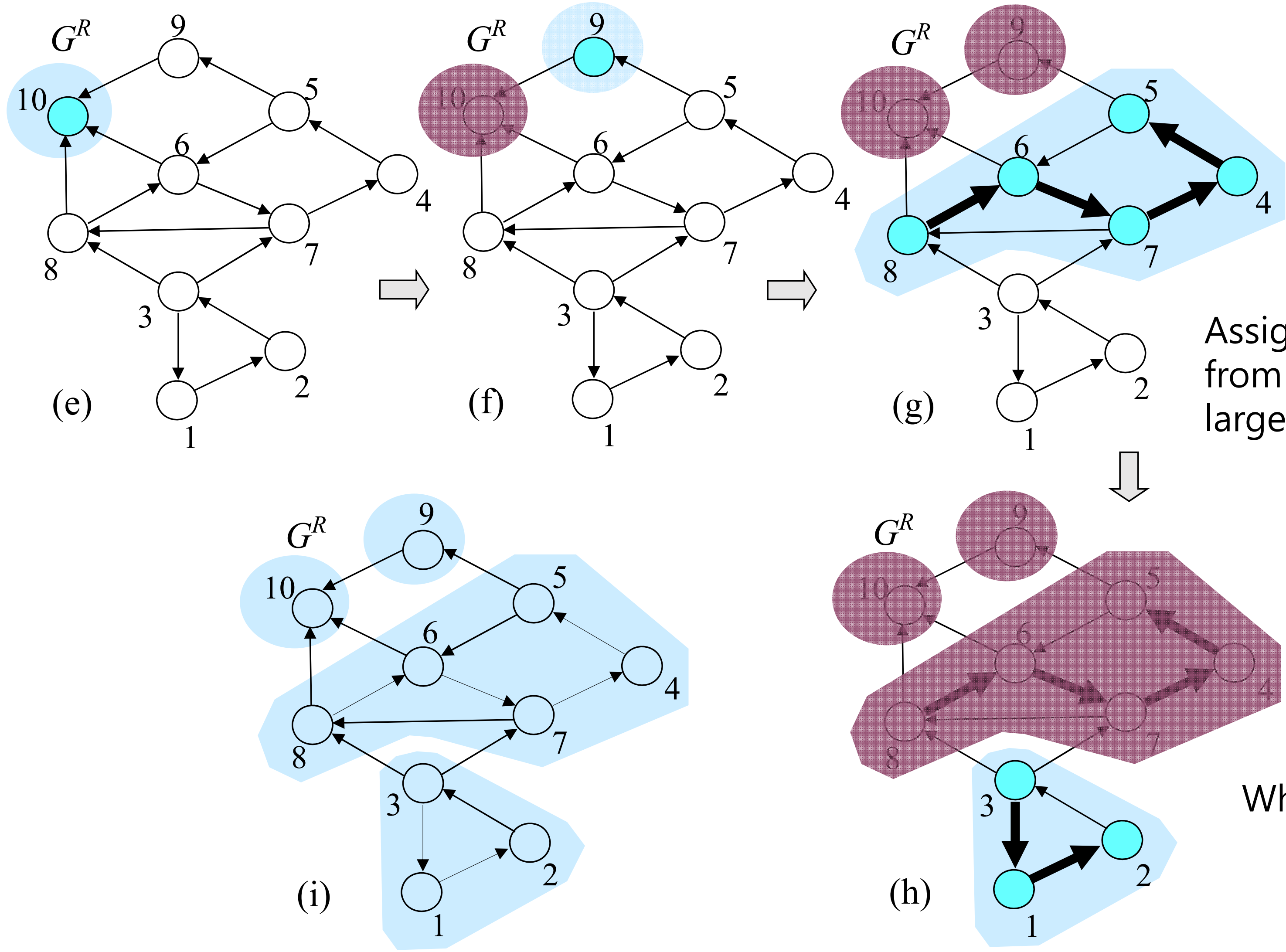
Kosaraju's algorithm



Search every nodes in DFS order.
Check visit or unvisit when we can confirm there are no nodes to search further.
Assign the order of visiting to each node
(Generation of G^R)

Reverse the direction of all edges





Assign all nodes reached from the unvisited largest node to an SCC

Why does it work?

Always there is a path from the largest node to the smallest node

In the graph of the reversed direction,
if there is a path from a node to a smaller node, then they are connected in both directions.

Creating SCC from the largest node can always guarantee that reachable nodes are smaller than a starting node

SCC 구하기 알고리즘

stronglyConnectedComponent(G)
{
1. 그래프 에 대해 DFS를 수행하여 각 정점 v의 완료시간 f_v를 계산한다;
2. G^R을 만든다;
3. DFS를 수행하되 시작점은 1에서 구한 f_v가 가장 큰 정점으로 잡는다;
4. 3에서 만들어진 분리된 트리들 각각을 강연결요소로 리턴한다;
}

✓수행시간: $\Theta(|V|+|E|)$