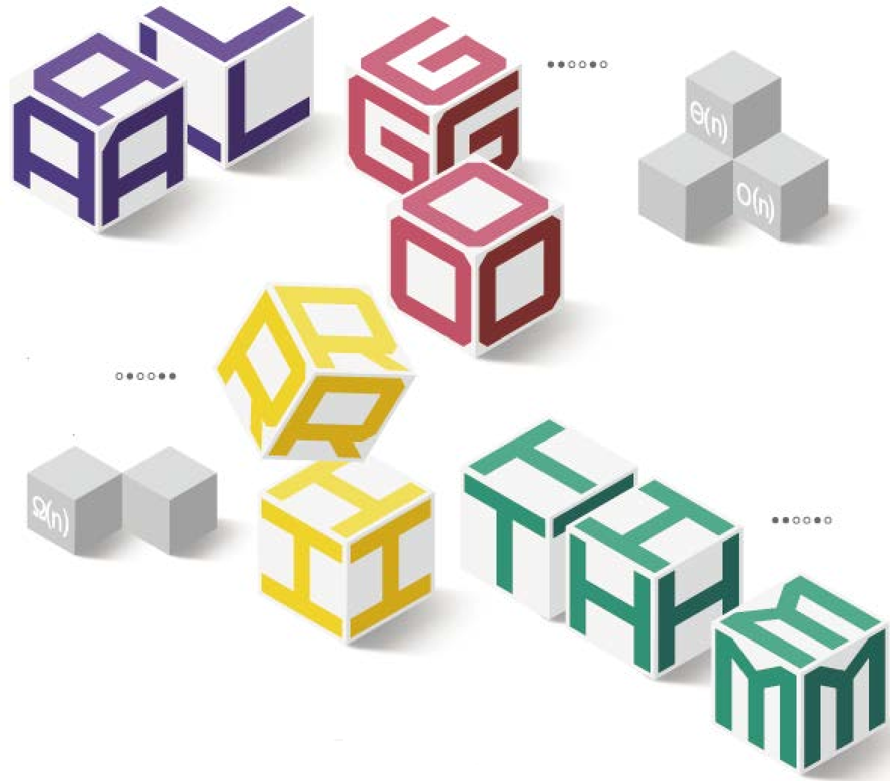


관계 중심의 사고법

쉽게 배우는 알고리즘

10장. 그래프 알고리즘

10장. 그래프 알고리즘

세상은 관계 투성이지.
"누가 누굴 좋아한다" 이런 것?
컴퓨터가 관계를 이용해서 무엇을 하려면 컴퓨터가 이해할 수 있는 관계의 표현이 앞서야겠지. 어떤 예가 있을까?
좋은 예. 어떤 고객이 어떤 상품을 샀다거나, 어떤 지점 간에 도로가 있다거나, 사건 A와 B는 인과 관계가 있다거나 이루 헤아릴 수 없이 많아.
이 장은 이런 그래프 표현으로 요리를 하는 알고리즘들을 배우지.

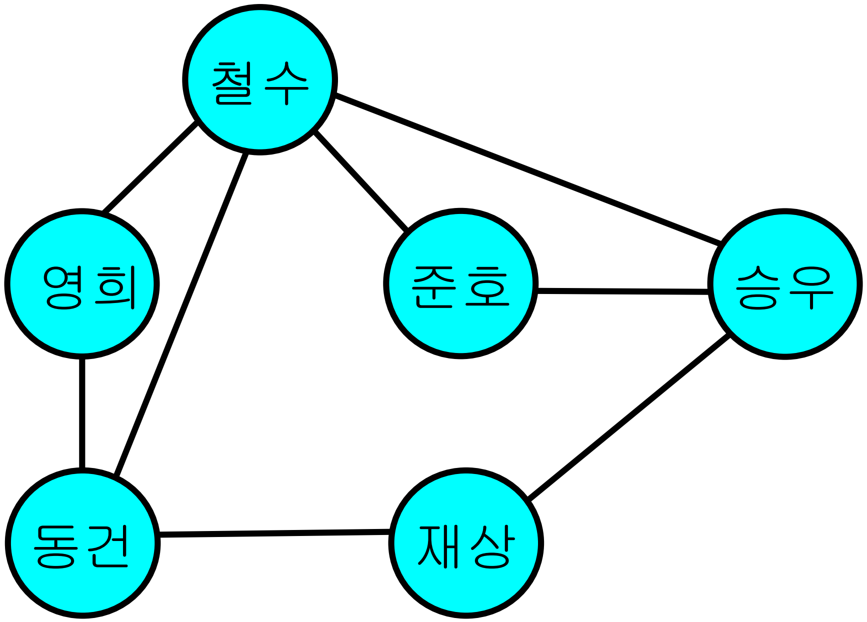
학습목표

- 그래프의 표현법을 익힌다.
- 너비 우선 탐색과 깊이 우선 탐색의 원리를 충분히 이해하도록 한다.
- 신장 트리의 의미와 최소 신장 트리를 구하는 두 가지 알고리즘을 이해한다.
- 그래프의 특성에 따라 가장 적합한 최단 경로 알고리즘을 선택할 수 있도록 한다.
- 위상 정렬을 알고, DAG의 경우에 위상 정렬을 이용해 최단 경로를 구하는 방법을 이해한다.
- 강연결 요소를 구하는 알고리즘을 이해하고 이 알고리즘의 정당성을 확신할 수 있도록 한다.
- 본문에서 소개하는 각 알고리즘의 수행 시간을 분석할 수 있도록 한다.

그래프

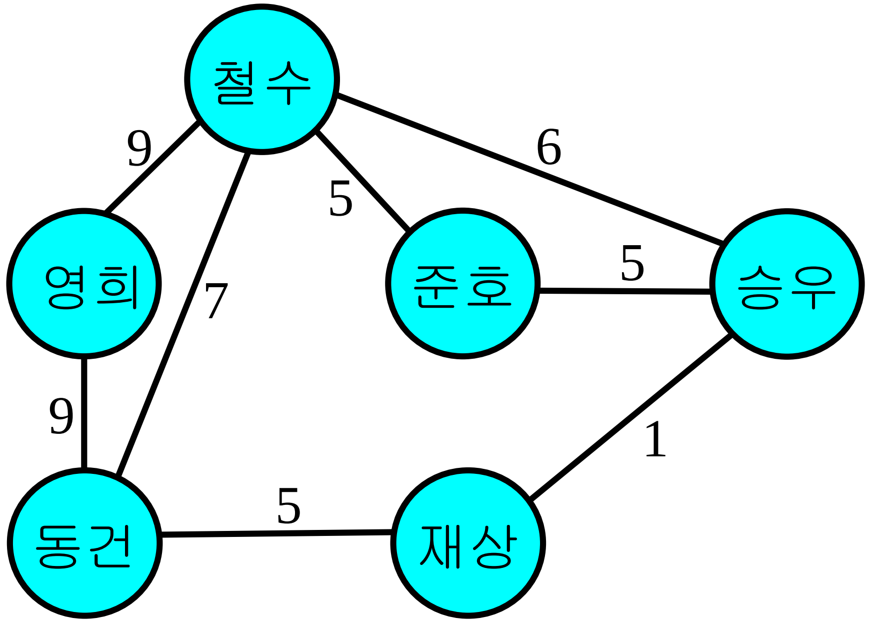
- 현상이나 사물을 정점vertex과 간선edge으로 표현한 것
- Graph $G = (V, E)$
  - V: 정점 집합
  - E: 간선 집합
- 두 정점이 간선으로 연결되어 있으면 인접adjacent하다고 한다
  - 간선은 두 정점의 관계를 나타낸다

그래프의 예



사람들간의 친분 관계를 나타낸 그래프

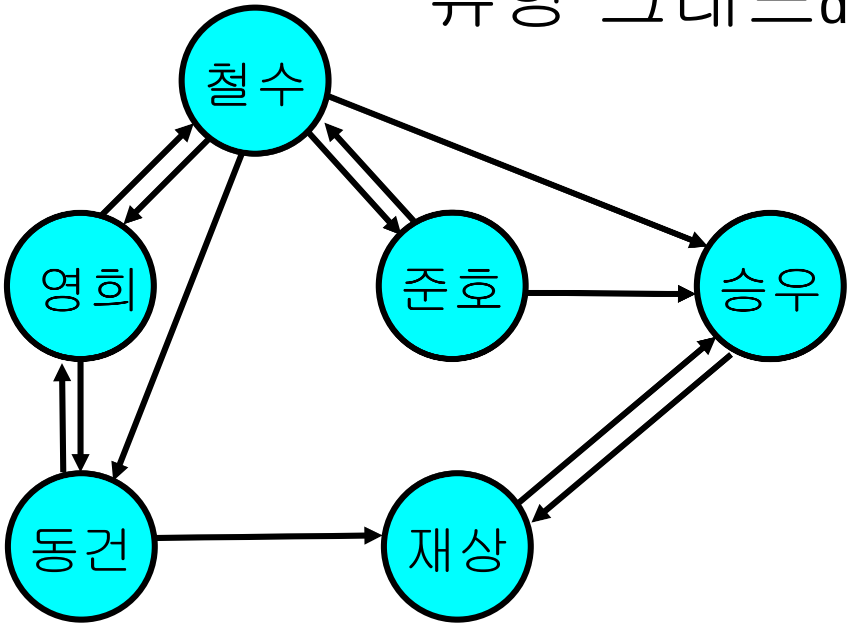
그래프의 예



친밀도를 가중치로 나타낸 친분관계 그래프

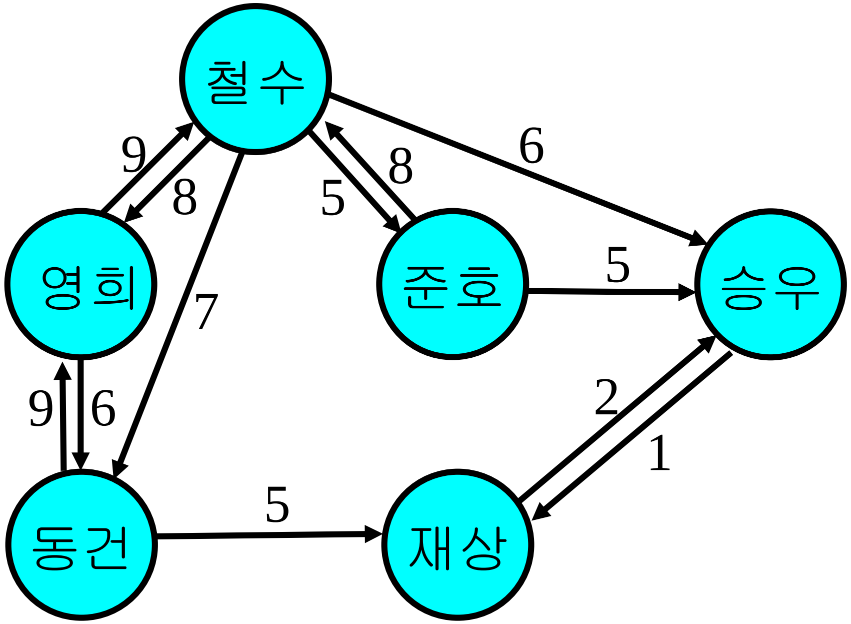
그래프의 예

유향 그래프directed graph=digraph



방향을 고려한 친분관계 그래프

그래프의 예



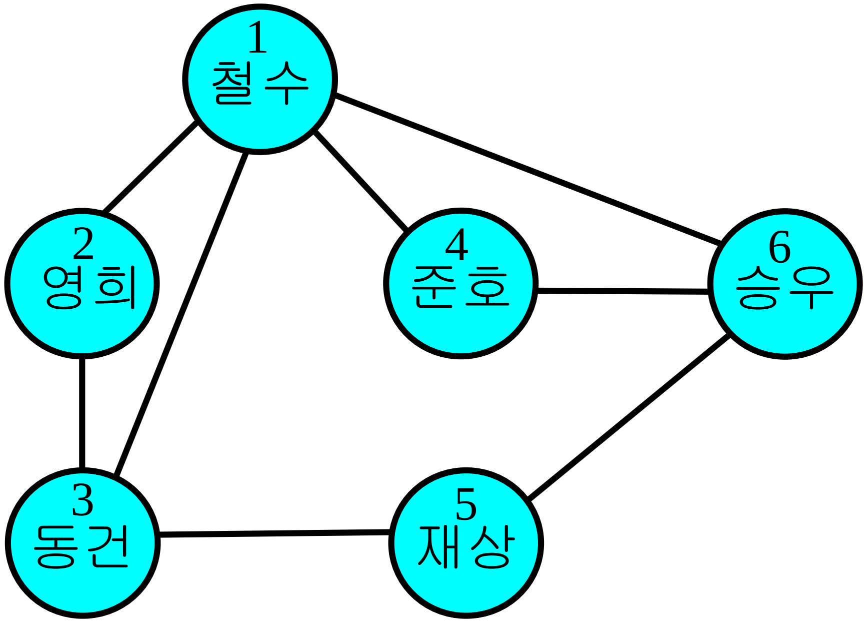
가중치를 가진 유향 그래프

그래프의 표현 1: 인접행렬

N: 정점의 총 수

- 인접행렬
  - $N \times N$ 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

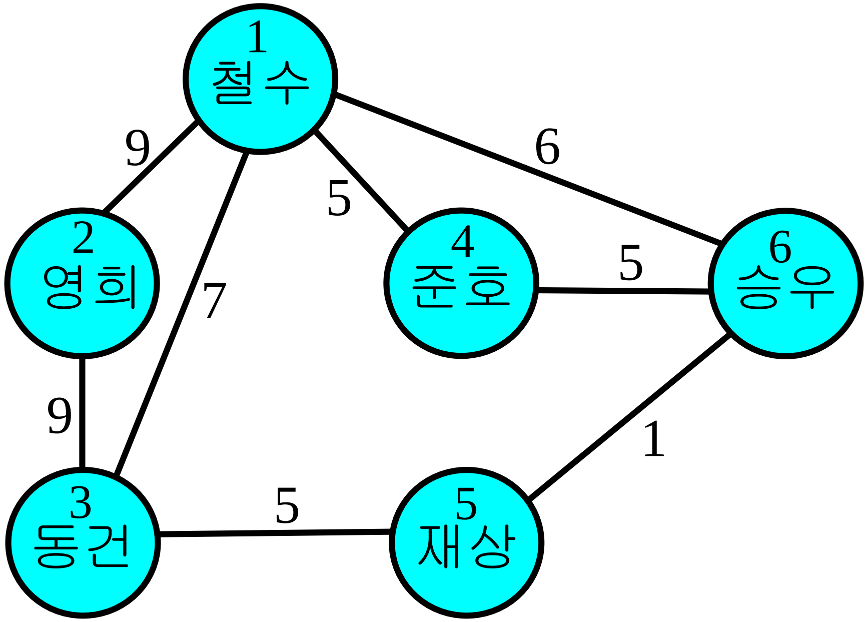
인접행렬



	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

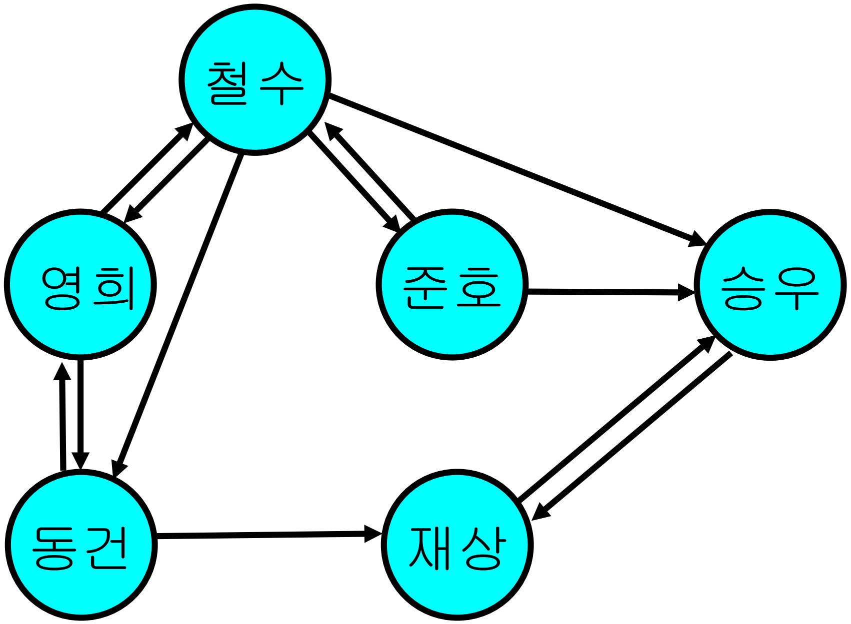
인접행렬



	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

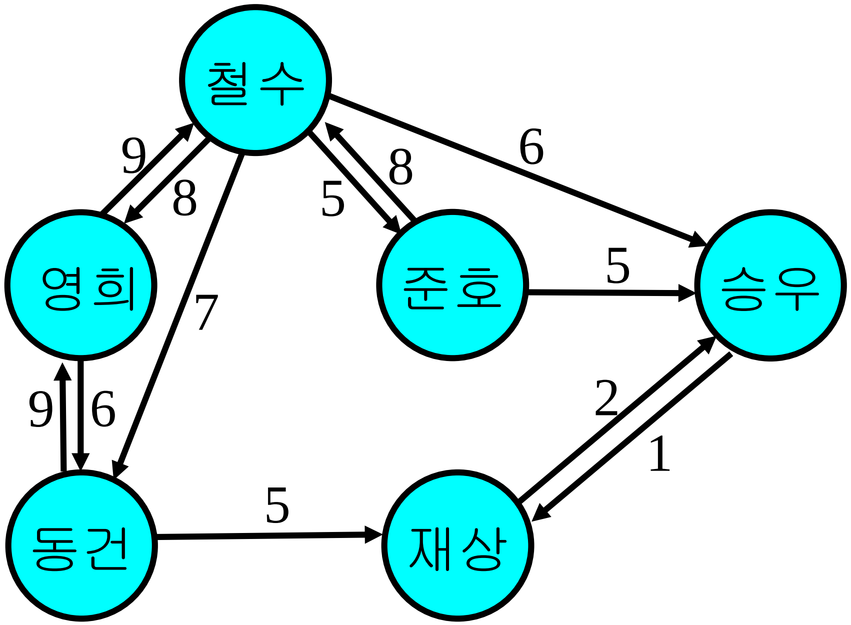
인접행렬



유향 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

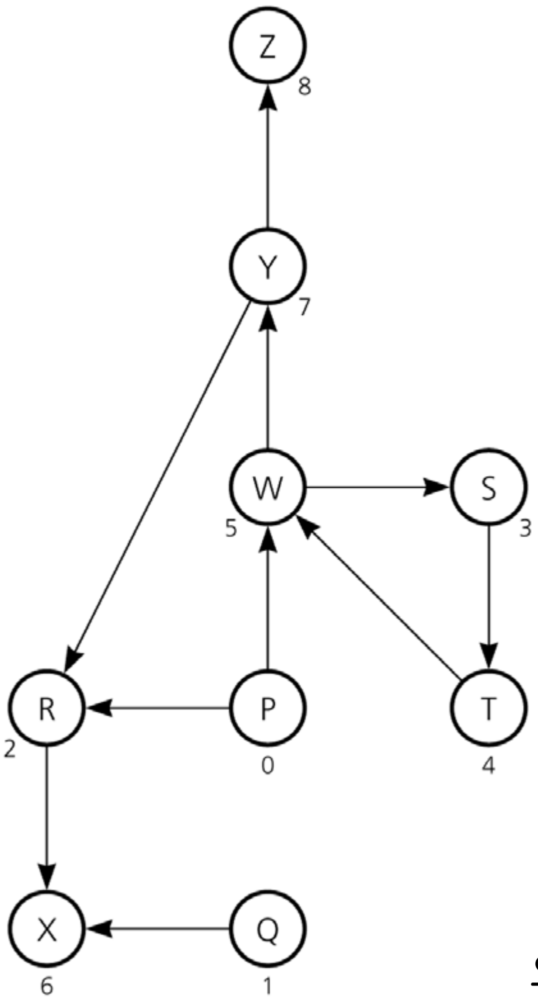
인접행렬



	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

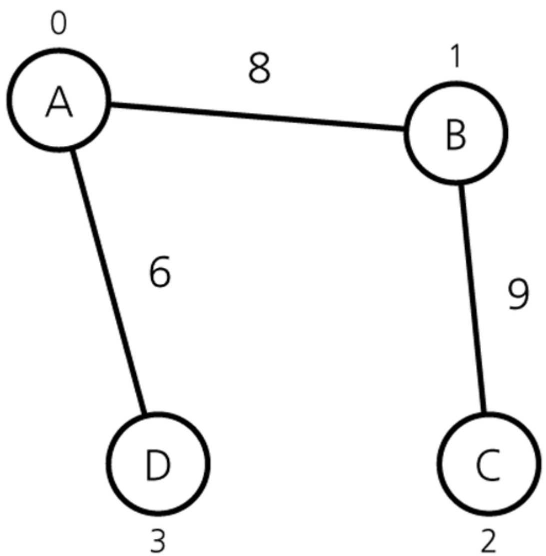
인접행렬



		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

인접행렬



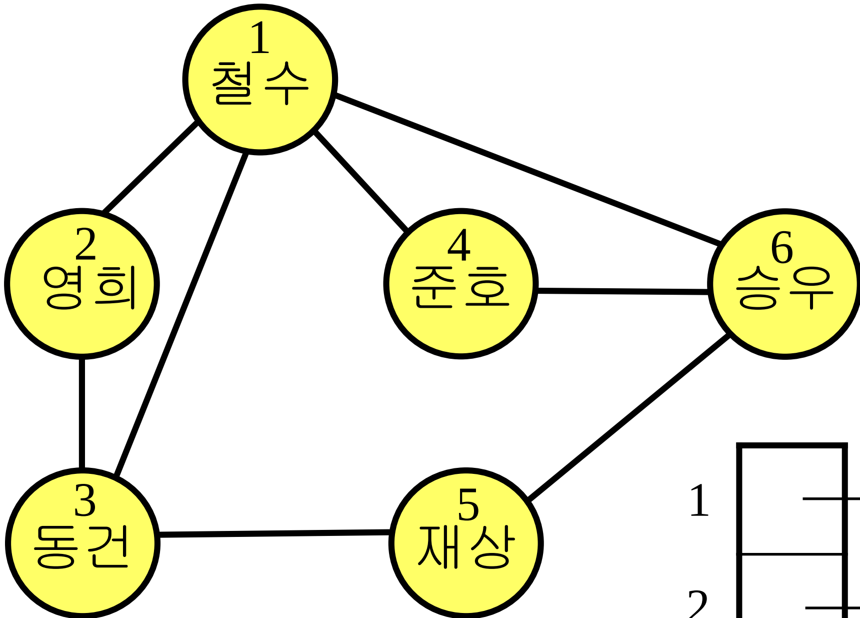
		0 A	1 B	2 C	3 D
0	A	∞	8	∞	6
1	B	8	∞	9	∞
2	C	∞	9	∞	∞
3	D	6	∞	∞	∞

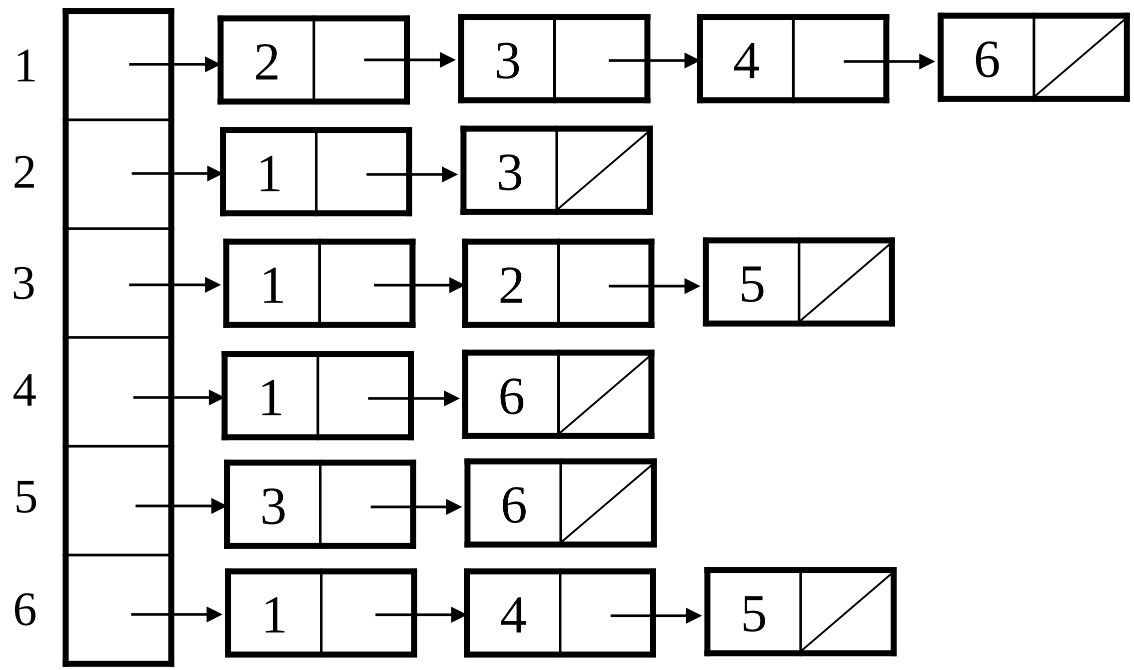
가중치 있는 그래프의 다른 예

그래프의 표현 2: 인접리스트

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

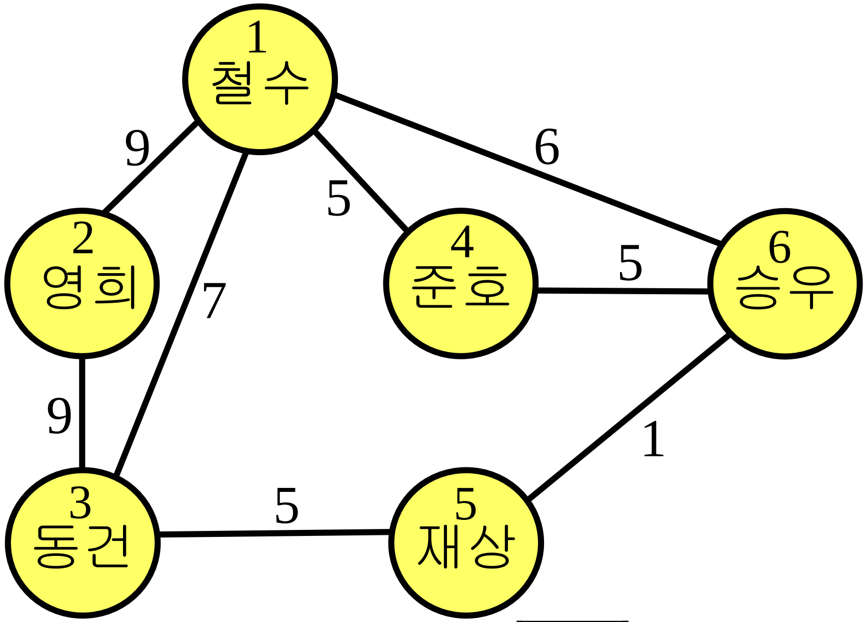
인접 리스트



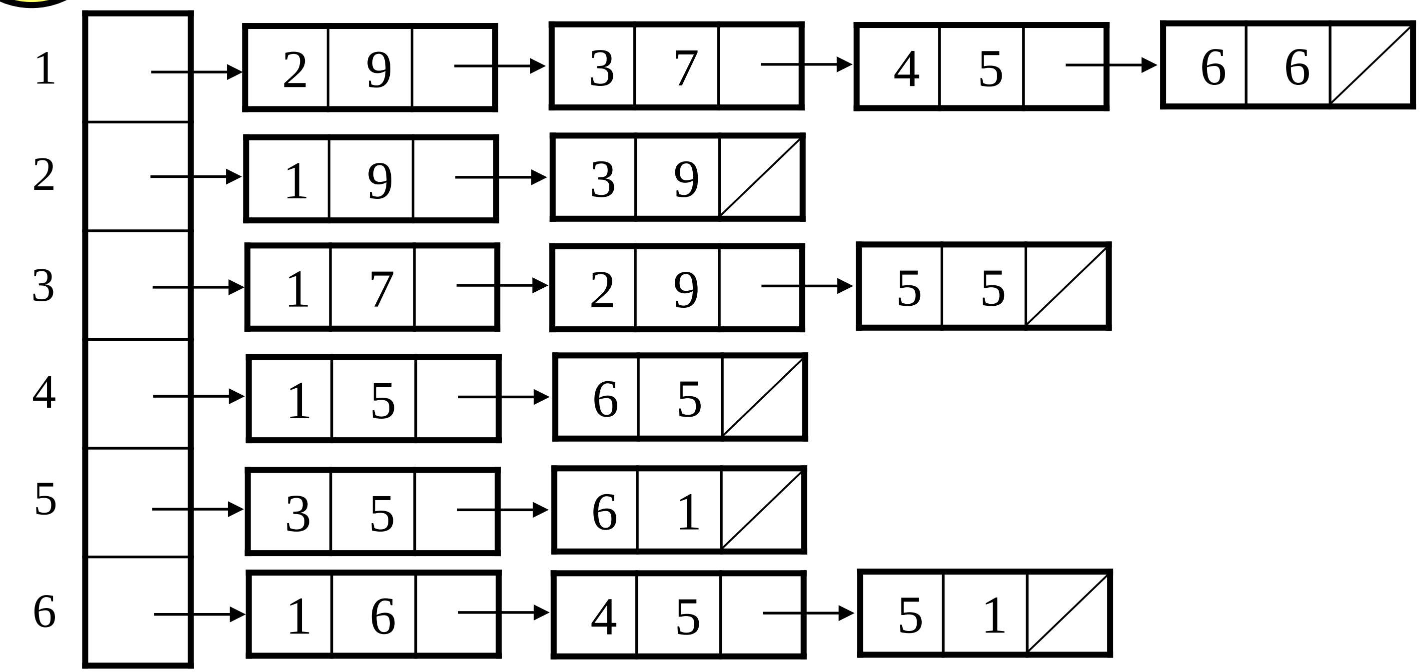


무향 그래프의 예

인접 리스트



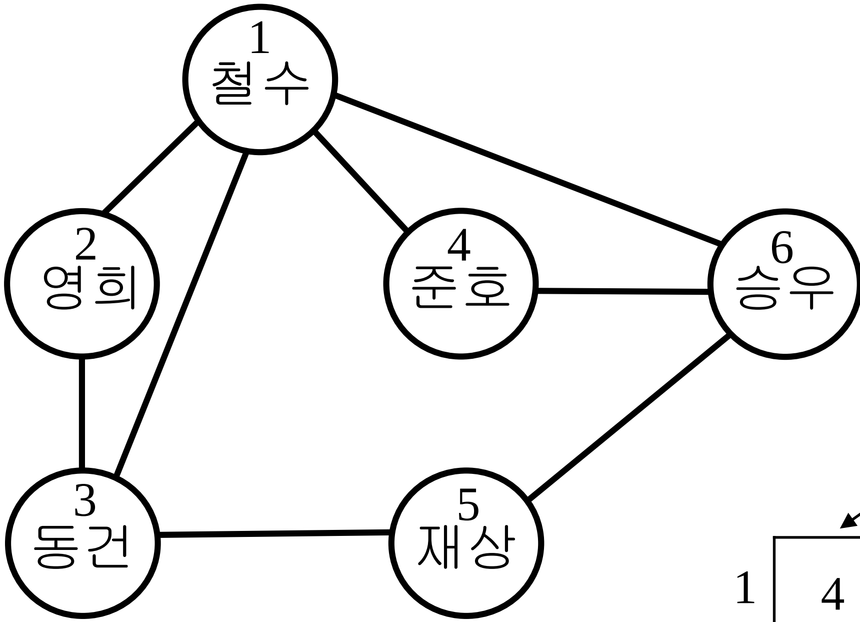
가중치 있는 그래프의 예

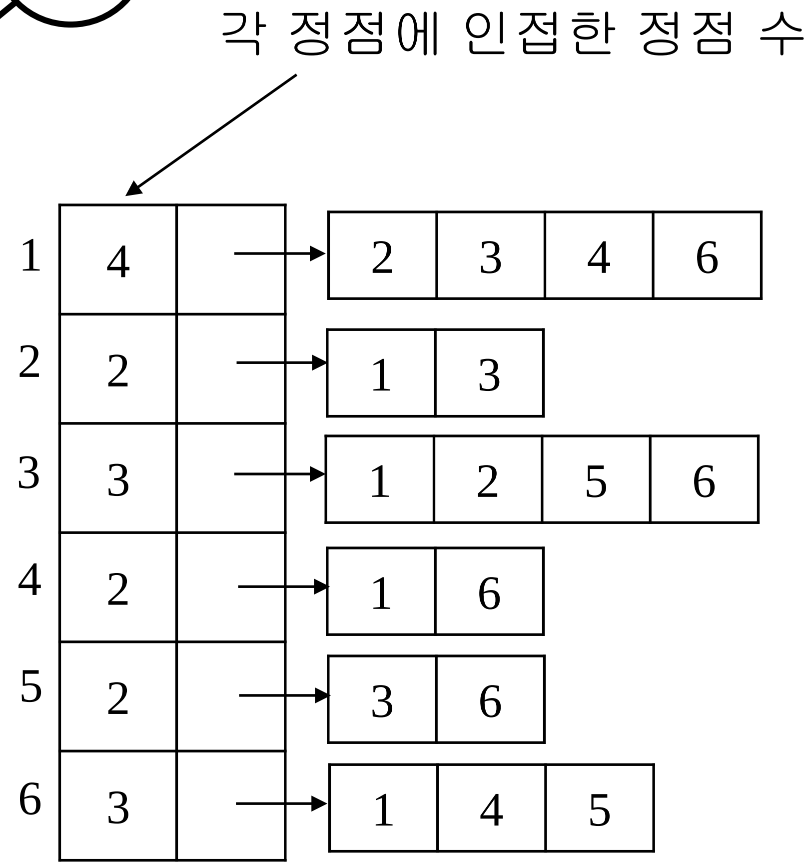


그래프의 표현 3: 인접 배열

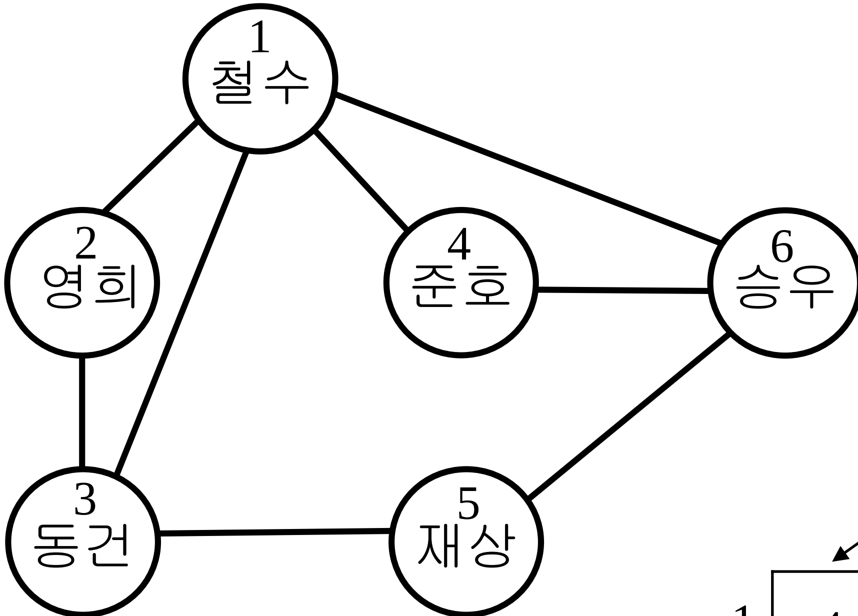
- 인접 배열
  - N 개의 연결 배열로 표현
  - – i 번째 배열은 정점 i 에 인접한 정점들을 집합
  - – 가중치 있는 그래프의 경우
    - 리스트에 가중치도 보관한다

인접 배열

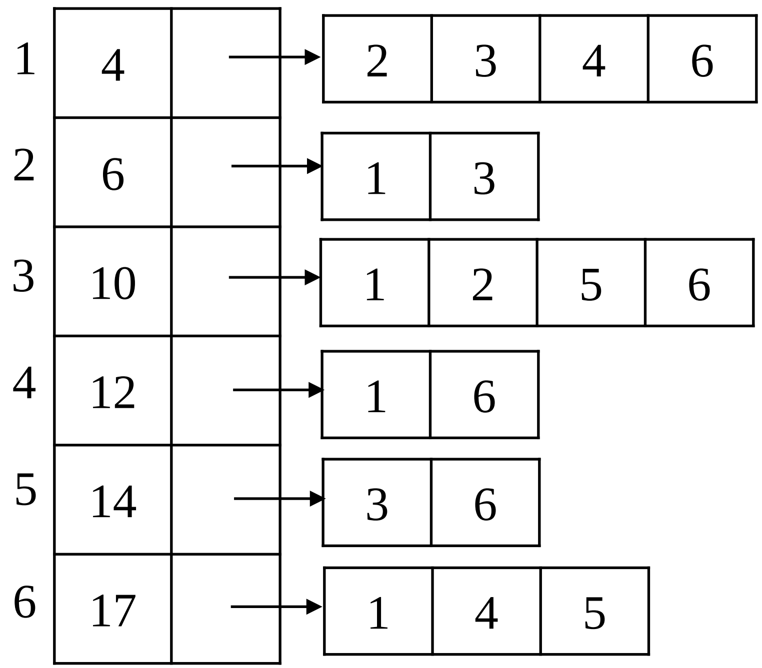

1 철수
2 영희
3 동건
4 준호
5 재상
6 승우


각 정점에 인접한 정점 수
1 4 2 3 4 6
2 2 1 3
3 3 1 2 5 6
4 2 1 6
5 2 3 6
6 3 1 4 5

인접 배열



배열에서 각 정점에 인접한
정점 목록의 끝자리



그래프에서 모든 정점 방문하기

- 대표적 두가지 방법
  - 너비우선탐색BFS (Breadth-First Search)
  - 깊이우선탐색DFS (Depth-First Search)
- 너무나 중요함
  - 그래프 알고리즘의 기본
  - DFS/BFS는 다 아는 듯이 보이지만 이해의 수준은 큰 차이가 난다
  - DFS/BFS는 뼛속 깊이 이해해야 좋은 그래프 알고리즘을 만들 수 있음

BFS너비우선탐색

```
BFS(G, v)
{
      for each v ∈ V –{s}
            visited[v] ← NO;
      visited[s] ← YES;                 ▷ s: 시작 정점
      enqueue(Q, s);                    ▷ Q: 큐
      while (Q ≠ ϕ) {
            u ← dequeue(Q);
            for each v ∈ L(u)   ▷ L(u): 정점 u의 인접 리스트
                  if (visited[v] = NO) then
                        visited[u] ← YES;
                        enqueue(Q, v);
                  }

      }
}
```

✓수행 시간: $\Theta(|V|+|E|)$

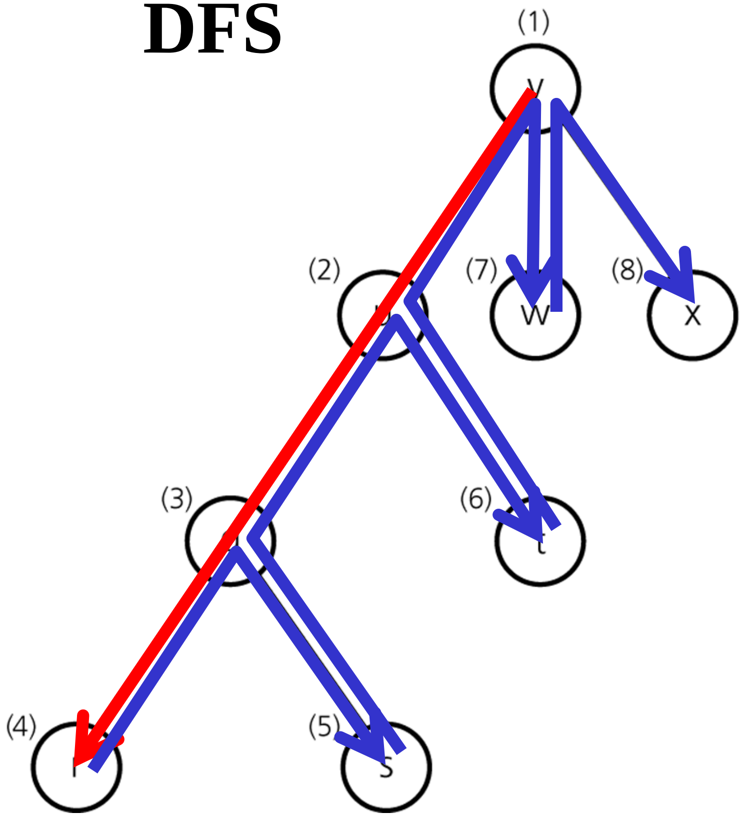
DFS깊이우선탐색

DFS(G)
{
    for each $v \in V$
        visited[v] $\leftarrow$ NO;
    for each $v \in V$
        if (visited[v] = NO) **then** aDFS(v);
}
aDFS (v)
{
    visited[v] $\leftarrow$ YES;
    for each $x \in L(v)$ $\triangleright$ $L(v)$: 정점 v의 인접 리스트
        if (visited[x] = NO) **then** aDFS(x);
}

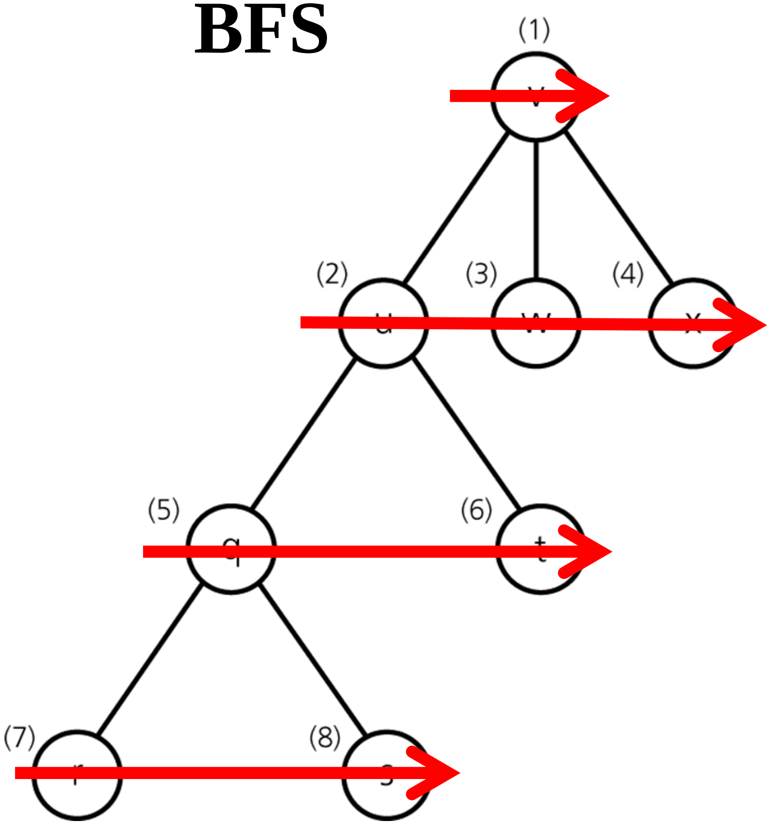
✓수행 시간: $\Theta(|V|+|E|)$

동일한 트리를 각각 DFS/BFS로 방문하기

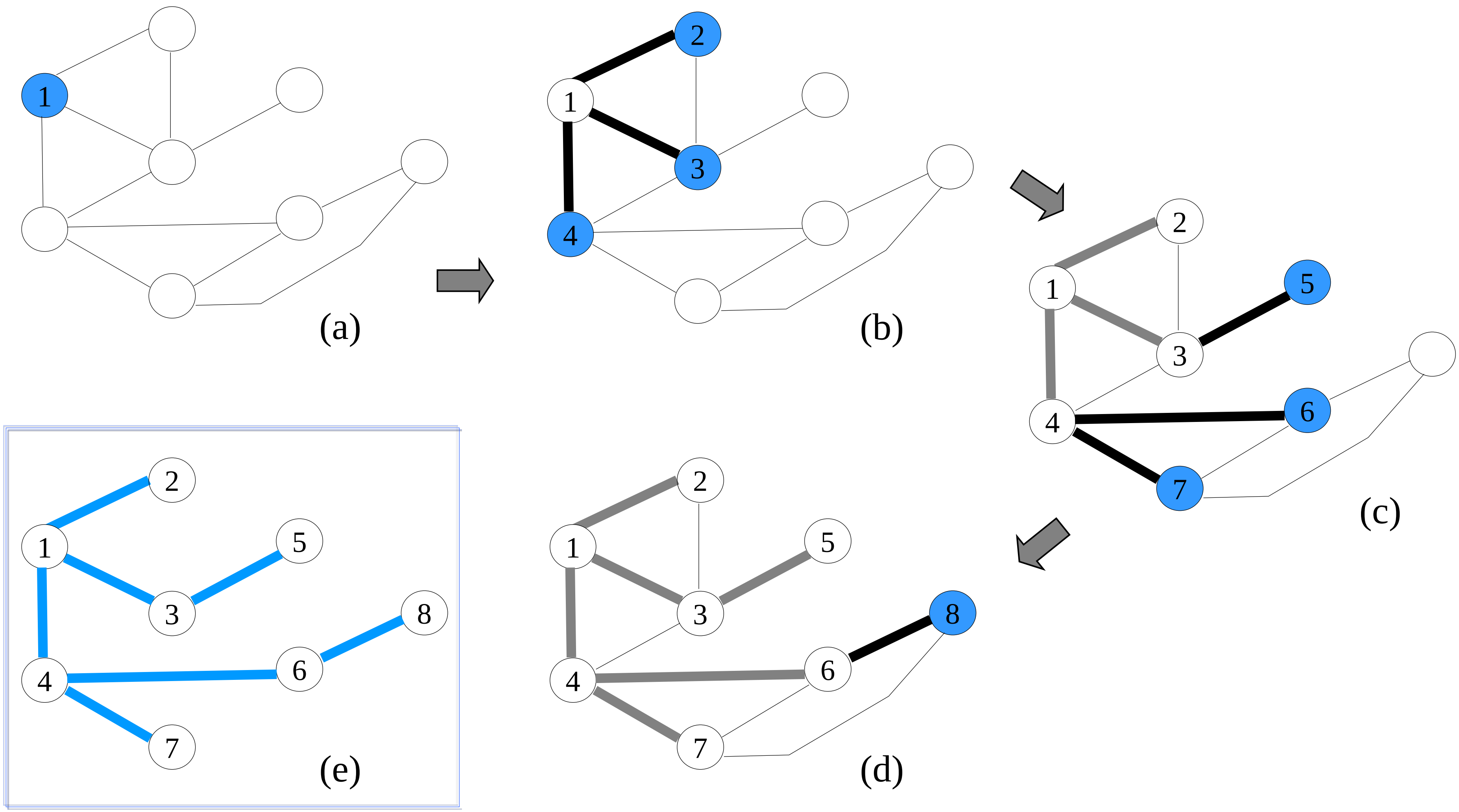
DFS



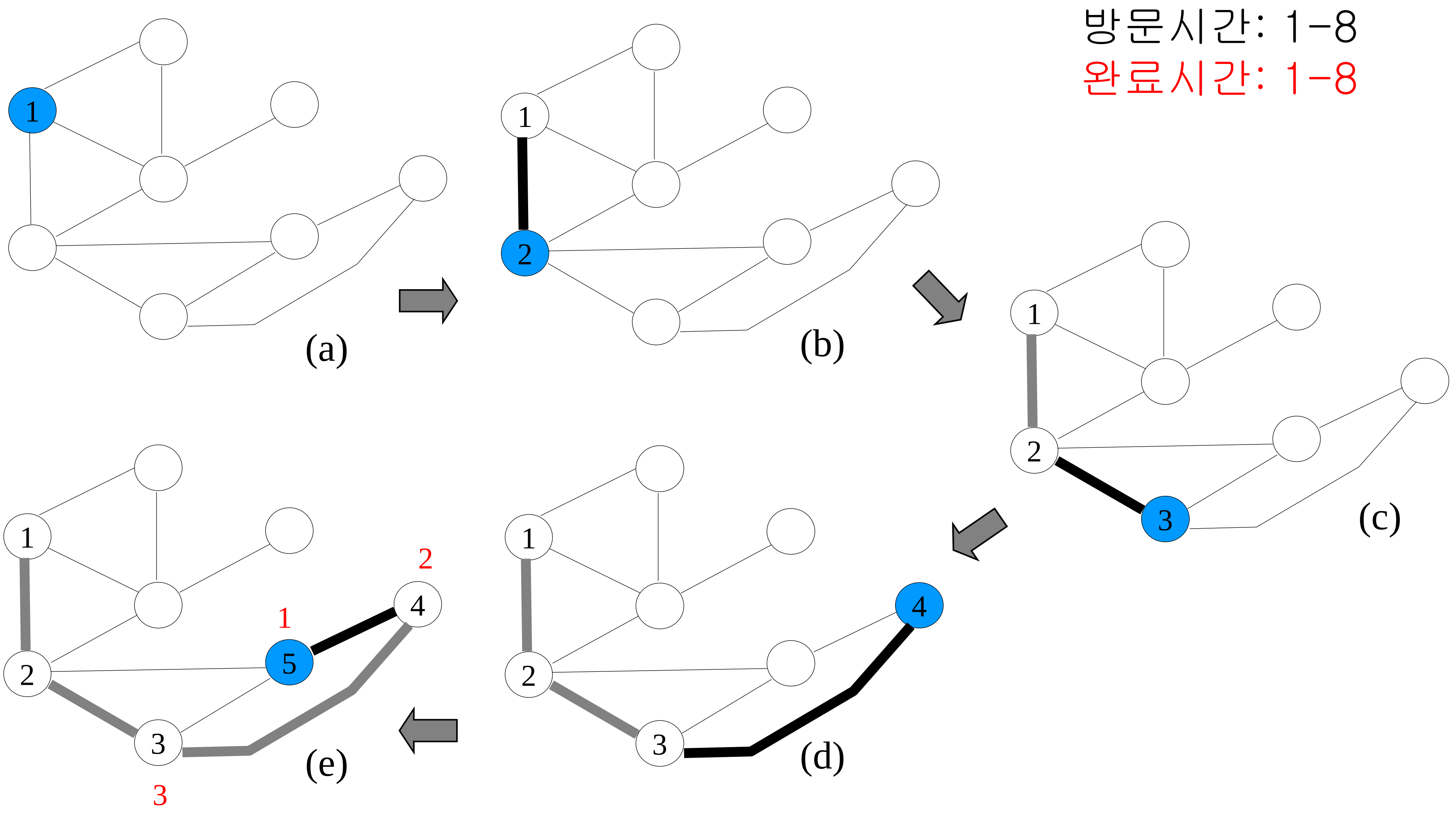
BFS



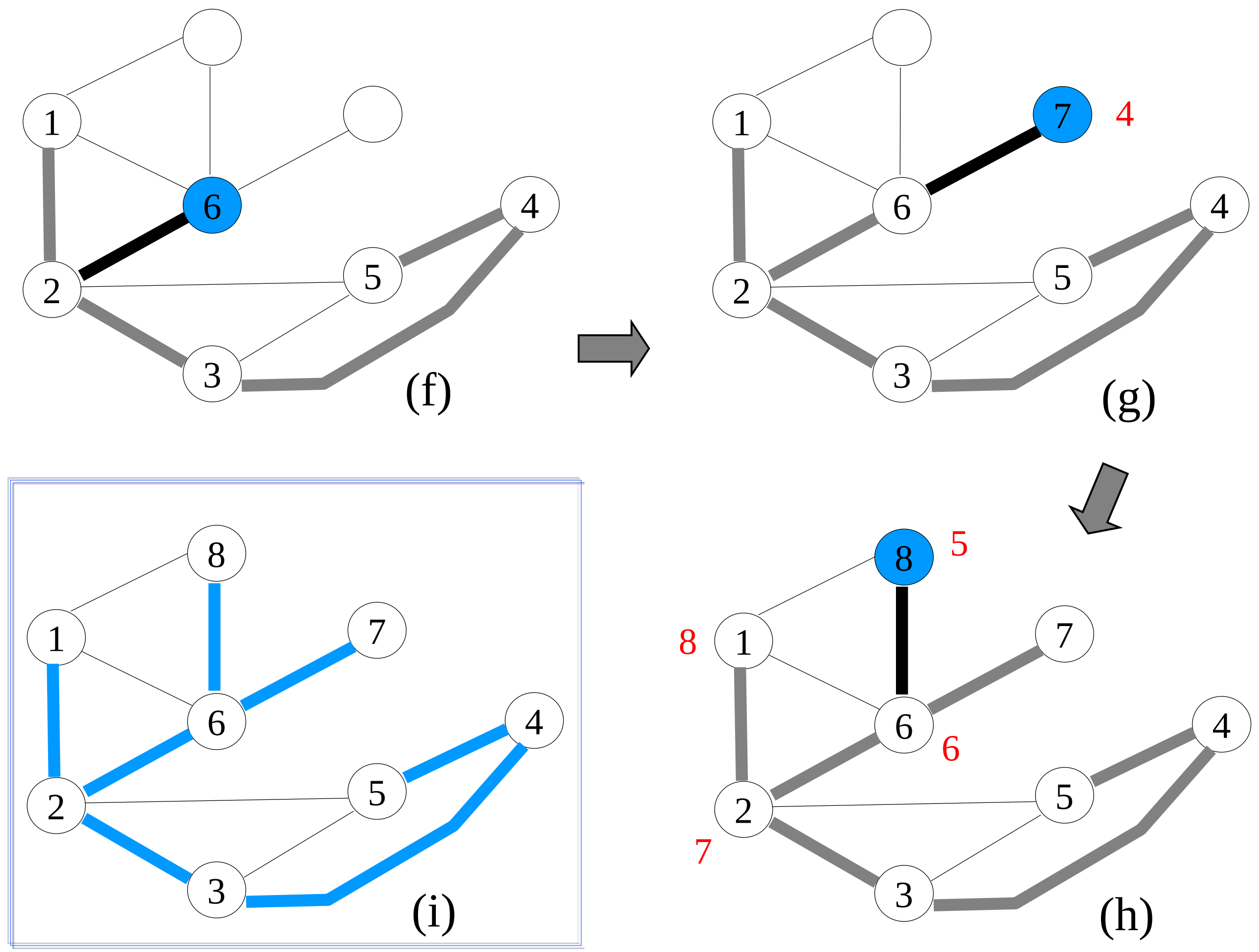
BFS의 작동 예



DFS의 작동 예

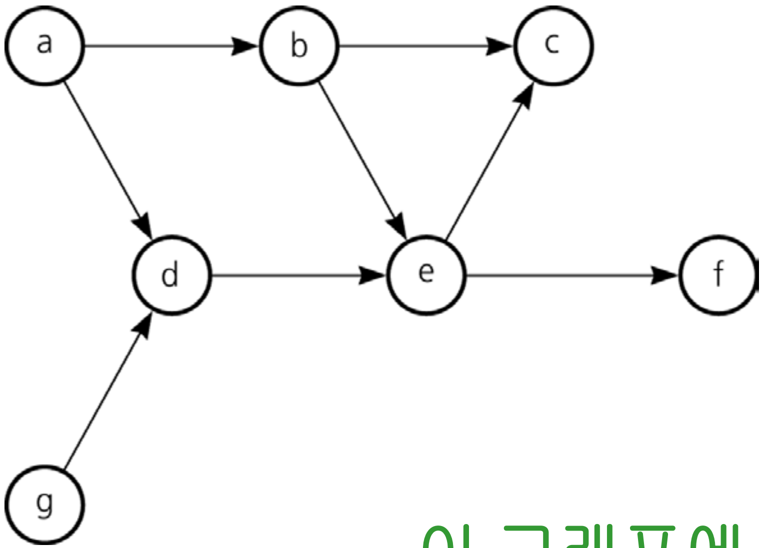


DFS의 작동 예 (계속)



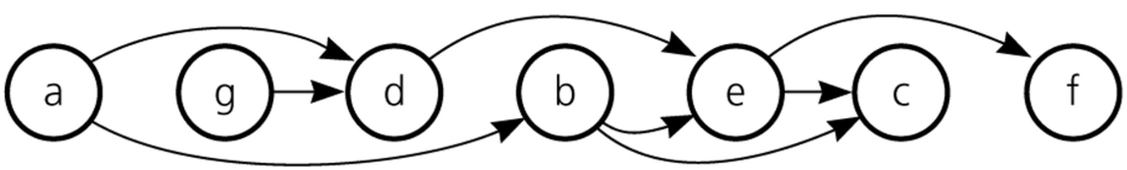
위상정렬 Topological sorting

- 조건
  - 싸이클이 없는 유향 그래프
- 위상정렬
  - 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

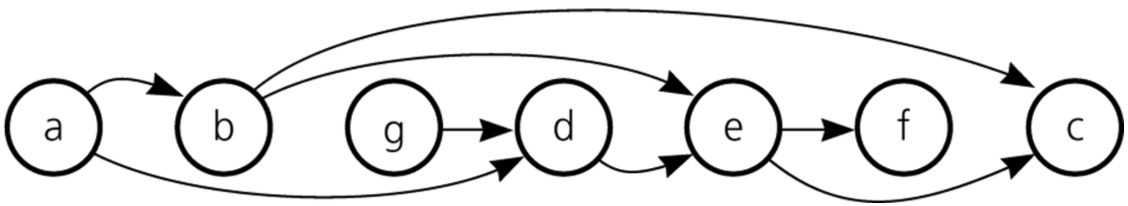


이 그래프에 대한 위상정렬의 예 2개

(a)



(b)

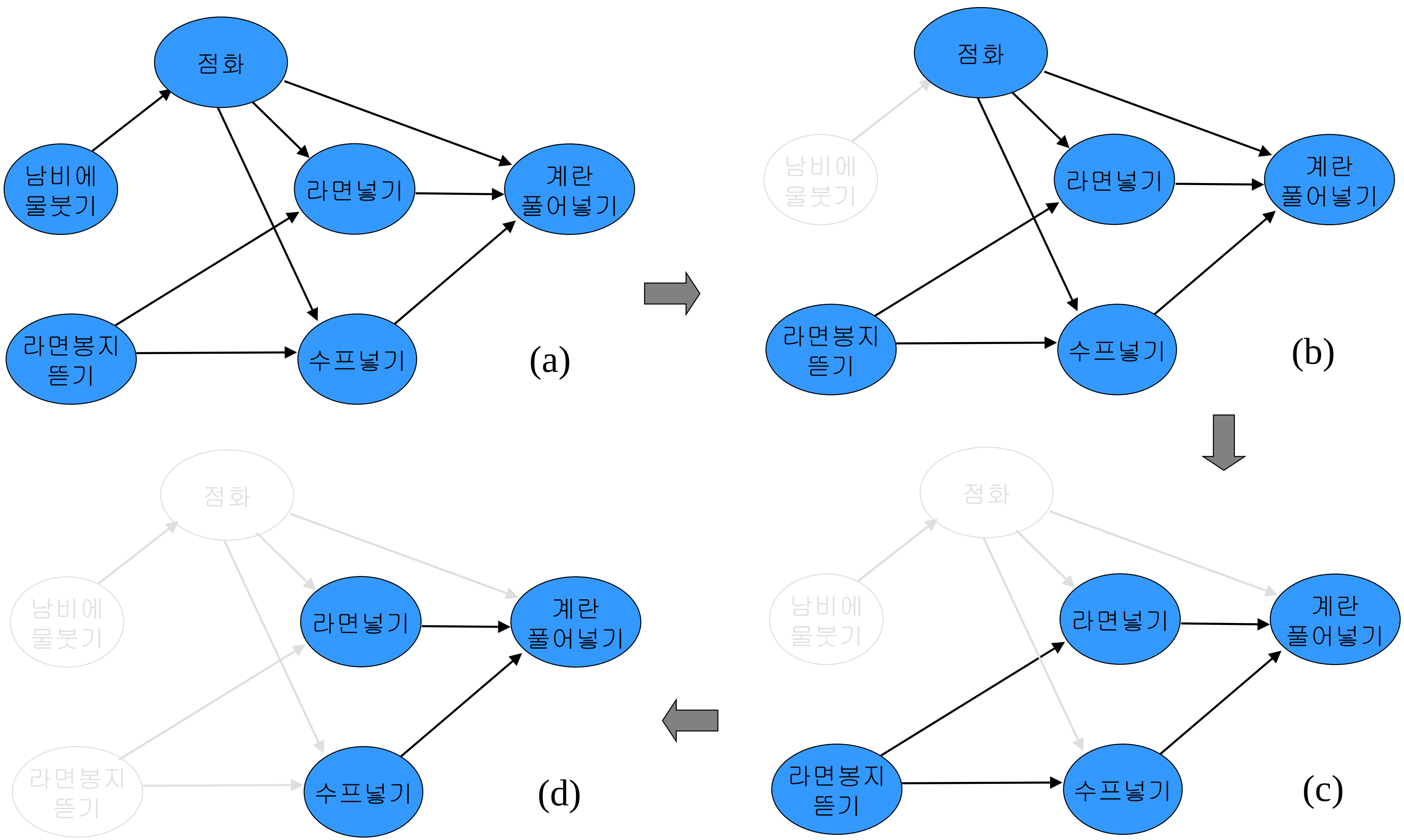


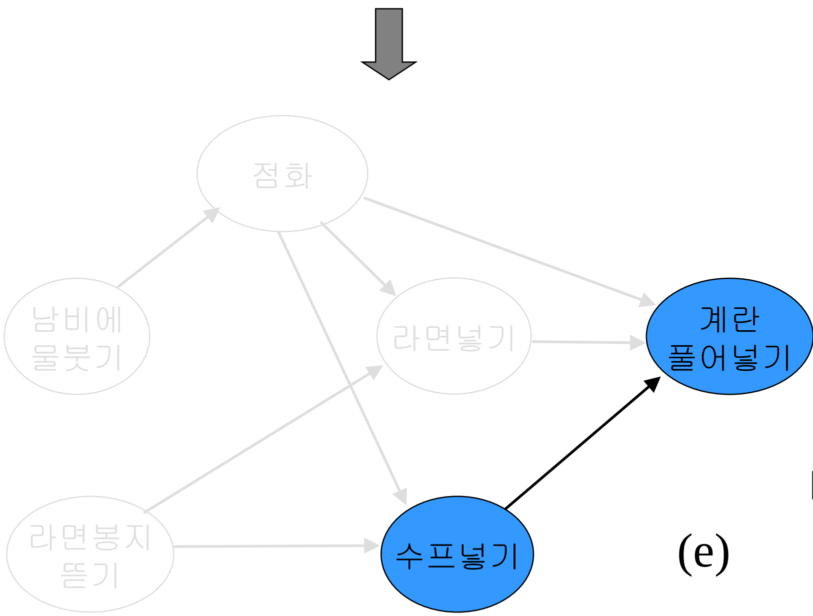
위상정렬 알고리즘 1

```
topologicalSort1(G, v)
{
    for ← 1 to n {
        진입간선이 없는 정점 u를 선택한다;
        A[i] ← u;
        정점 u와, u의 진출간선을 모두 제거한다;
    }
    ▷ 이 시점에 배열 A[1…n]에는 정점들이 위상정렬되어 있다
}
```

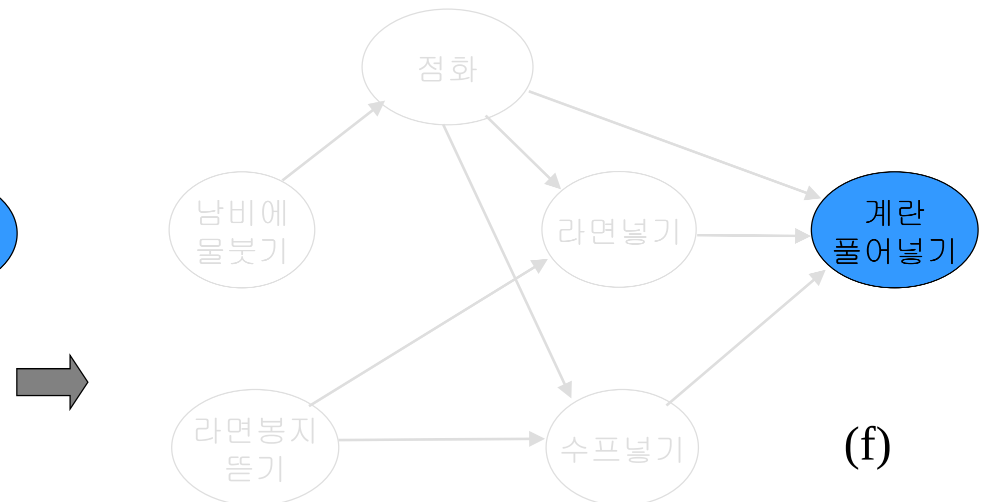
✓수행 시간: $\Theta(|V|+|E|)$

위상정렬 알고리즘 1의 작동 예

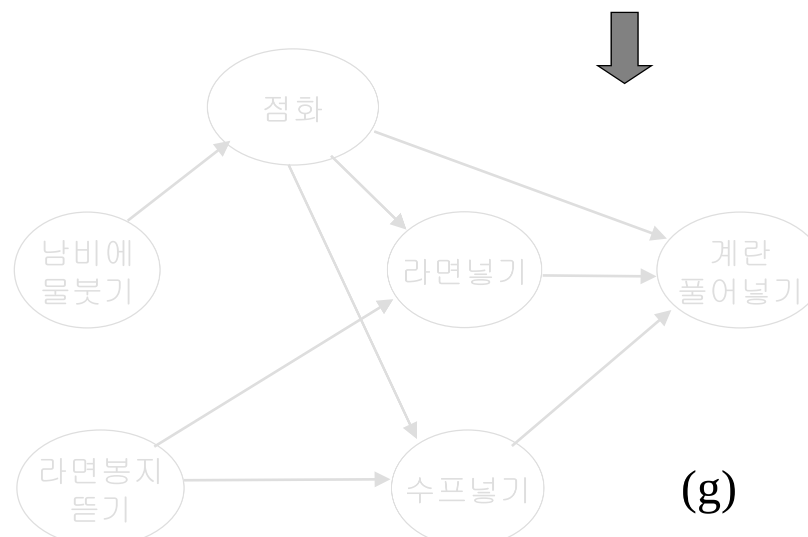



점화
남비에 물붓기
라면넣기
계란 풀어넣기
라면봉지 뜯기
수프넣기

(e)


점화
남비에 물붓기
라면넣기
계란 풀어넣기
라면봉지 뜯기
수프넣기

(f)


점화
남비에 물붓기
라면넣기
계란 풀어넣기
라면봉지 뜯기
수프넣기

(g)

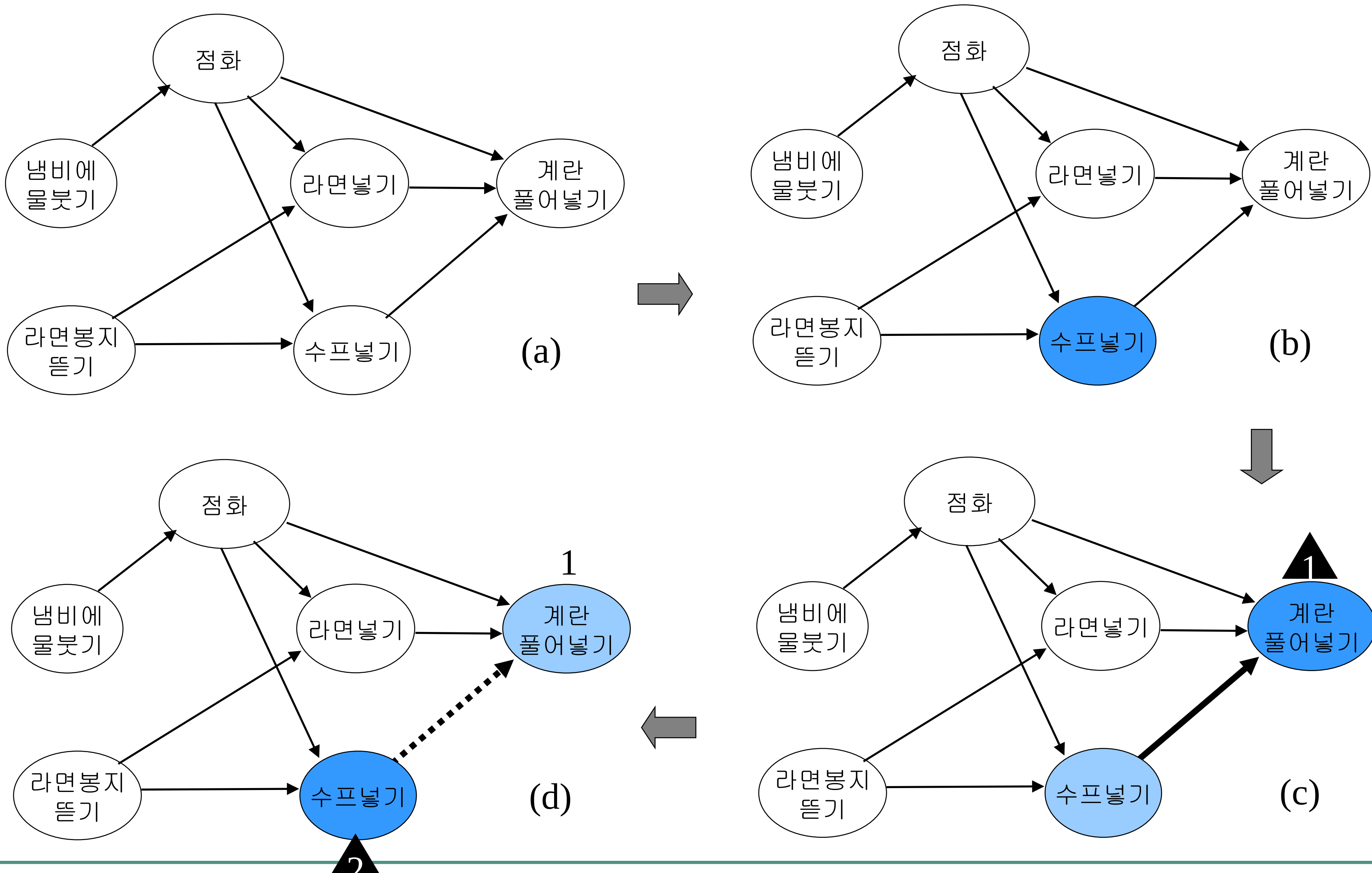
위상정렬 알고리즘 2

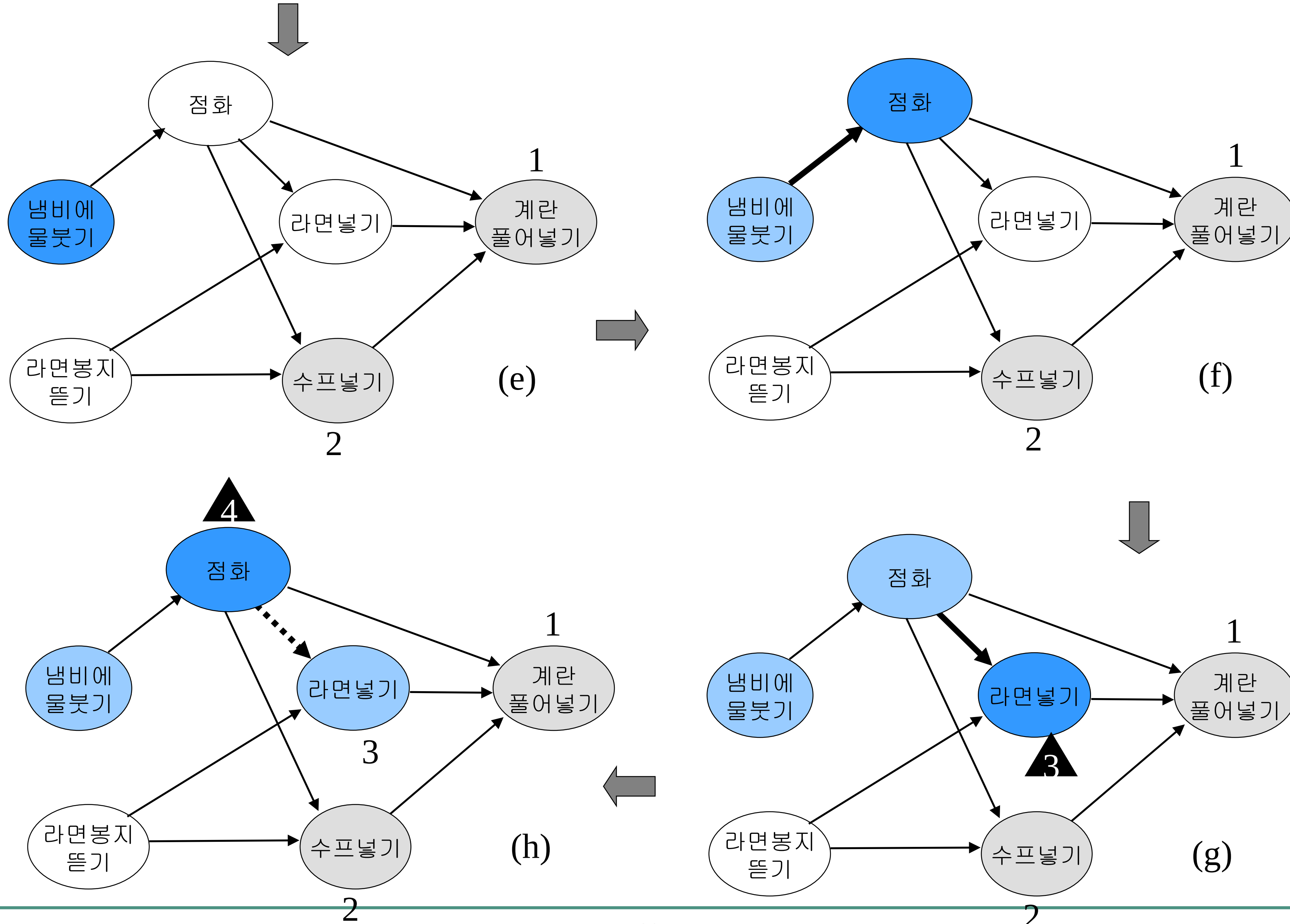
```
topologicalSort2(G)
{
    for each v∈V
        visited[v] ← NO;
    for each v∈V    ▷ 정점의 순서는 무관
        if (visited[v] = NO) then DFS-TS(v) ;
}
DFS-TS(v)
{
    visited[v] ← YES;
    for each x∈L(v)    ▷ L(v): v의 인접 리스트
        if (visited[x] = NO) then DFS-TS(x) ;
    연결 리스트 R의 맨 앞에 정점 v를 삽입한다;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

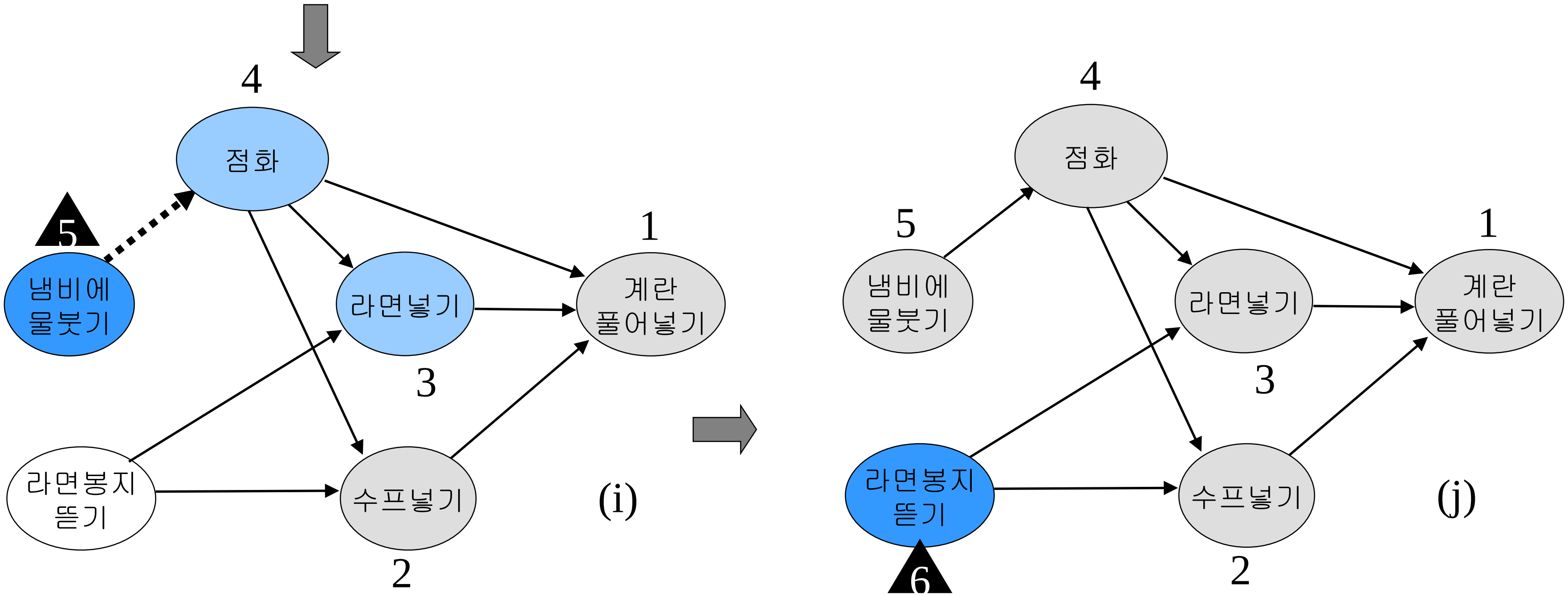
✓수행 시간: $\Theta(|V|+|E|)$

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

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


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

최단경로Shortest 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

다익스트라Dijkstra 알고리즘

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[u] +w[u, v] < d[v] ) then {
                    d[v]  ←  d[u] + w[u, v];
                    prev[v]  ←  u;
    }
}
```

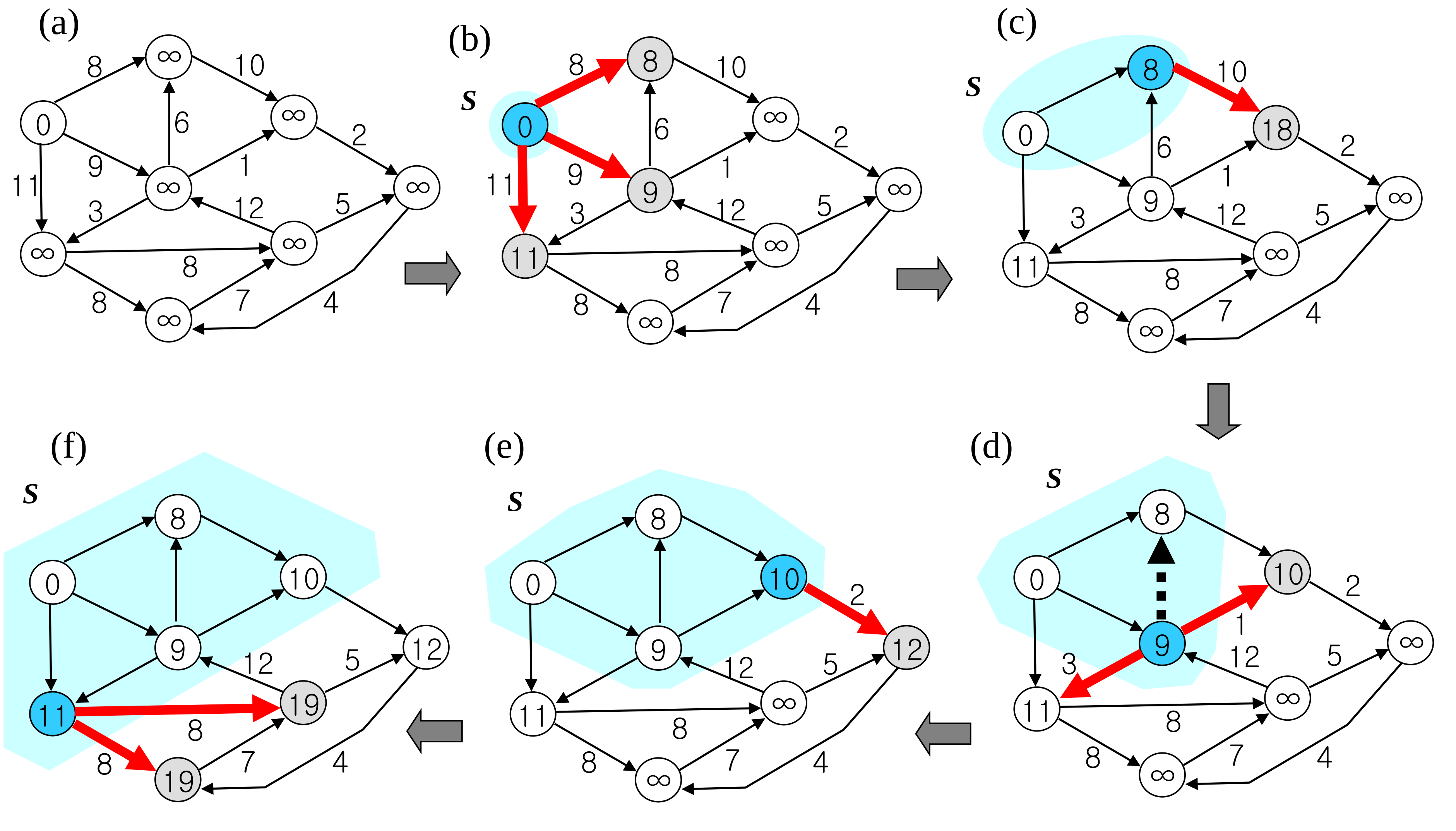
이완(relaxation)

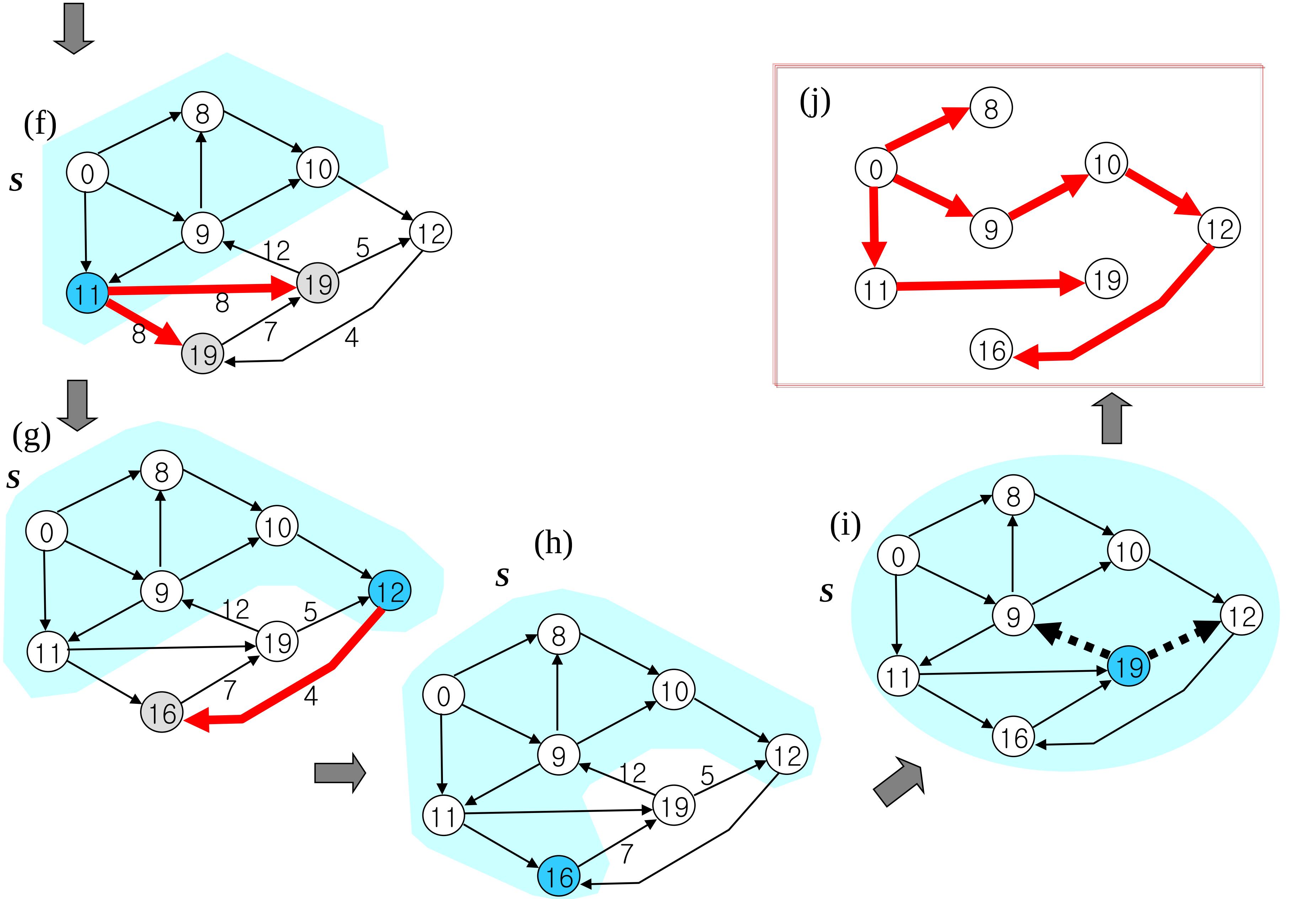
```
extractMin(Q, d[])
{
    집합 Q에서 d값이 가장 작은 정점 u를 리턴한다 ;
}
```

✓수행 시간: $O(|E|\log|V|)$

힙 이용

다익스트라 알고리즘의 작동 예




(f)
s
0
8
9
10
11
12
19
19
12
5
8
8
7
4
(g)
s
0
8
9
10
11
12
16
19
12
5
7
4
(h)
s
0
8
9
10
11
12
16
19
12
5
7
(i)
s
0
8
9
10
11
12
16
19
(j)
0
8
9
10
11
12
16
19

벨만-포드Bellman-Ford 알고리즘

음의 가중치를 허용한다

```
BellmanFord(G, r)
{
    for each u∈V
            d[u] ← ∞;
    d[r] ← 0;
    for i ← 1 to |V|-1
            for each (u, v) ∈E
                    if (d[u] + w[u, v] < d[v]) then {
                            d[v] ← d[u] + w[u, v] ;
                            prev[v] ← u;
                    }
    ▷ 음의 싸이클 존재 여부 확인
    for each (u, v) ∈E
            if (d[u] + w[u, v] < d[v]) output "해없음";
}
```

이완(relaxation)

✓수행 시간: $\Theta(|E||V|)$

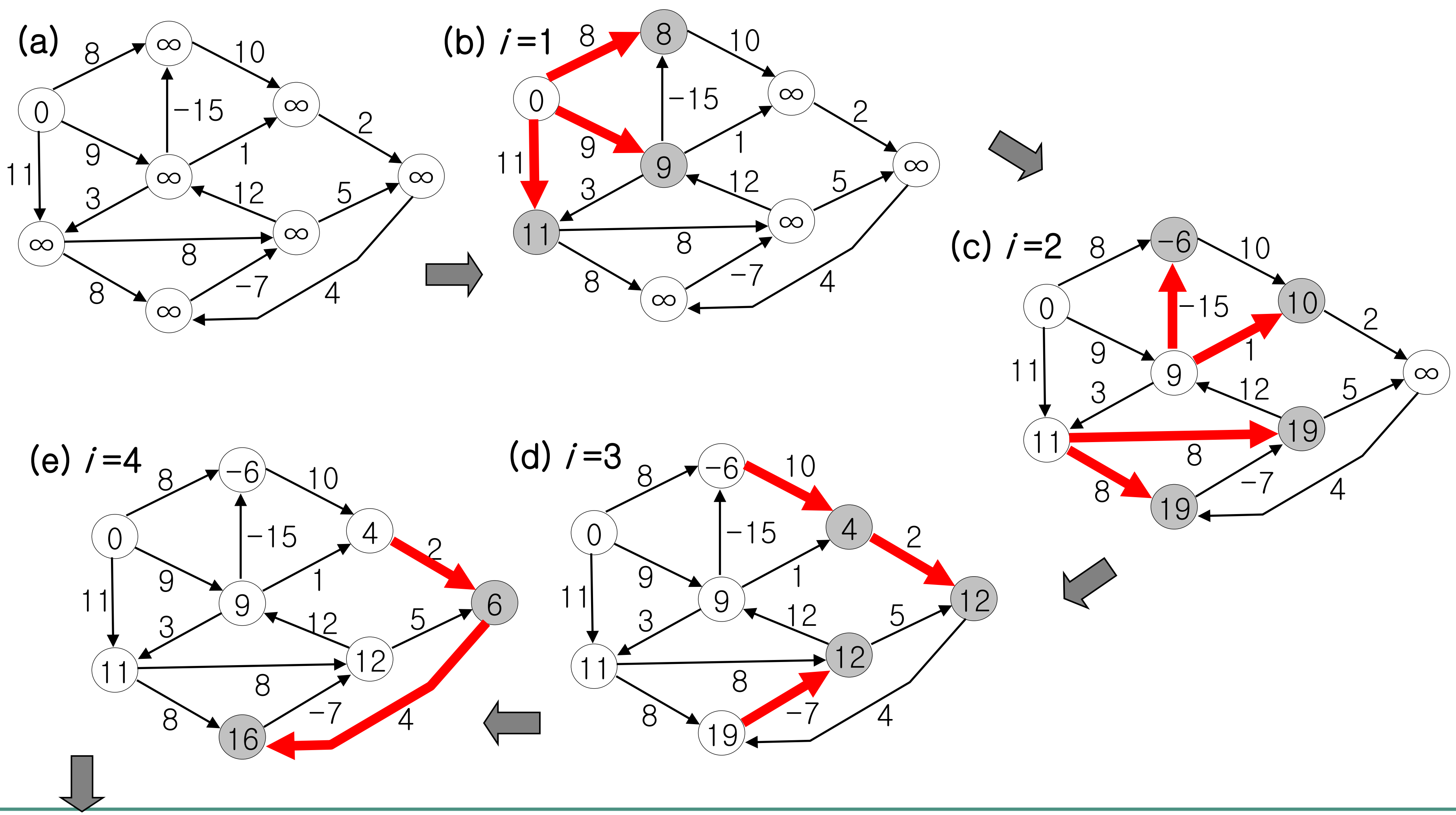
동적 프로그래밍으로 본 벨만-포드 알고리즘

- d_t^k : 중간에 최대 k 개의 간선를 거쳐
  정점 r로부터 정점 t에 이르는 최단거리
- 목표: d_t^{n-1}

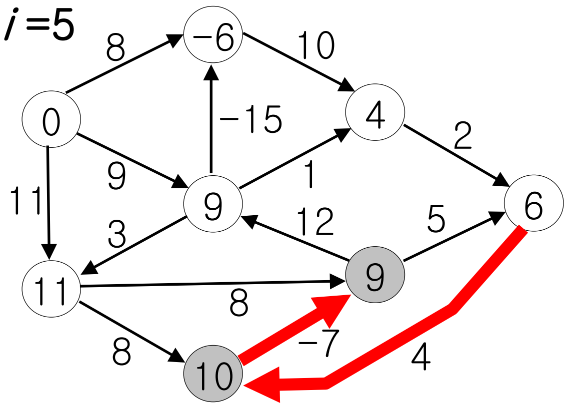
✓ 재귀적 관계

$$
\begin{cases}
d_v^k = \min\limits_{\text{for 모든 간선 } (u,\, v)} \{d_u^{k-1} + w_{uv}\}, & k > 0 \\
d_r^0 = 0 & \\
d_t^0 = \infty, & t \neq r
\end{cases}
$$

벨만-포드 알고리즘의 작동 예

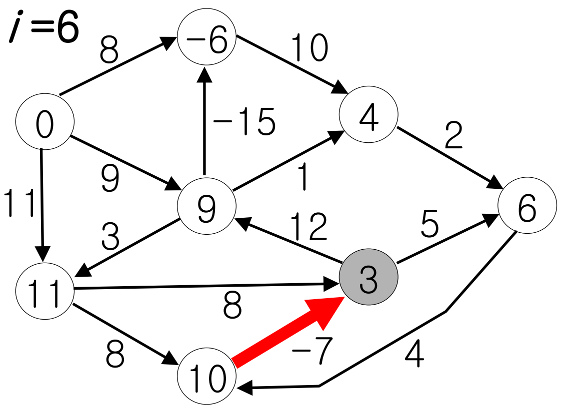


(f) $i = 5$



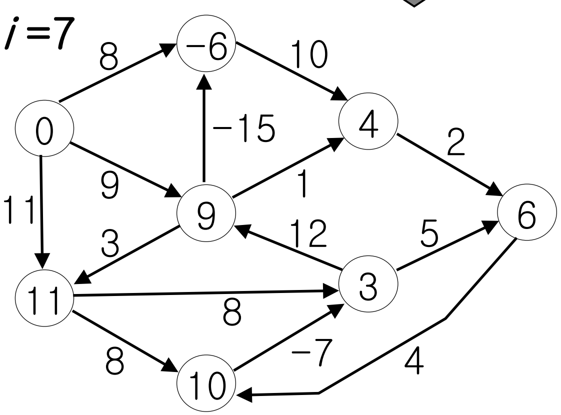
0
−6
4
9
6
11
9
10
8
10
−15
2
9
1
11
3
12
5
8
8
−7
4

(g) $i = 6$



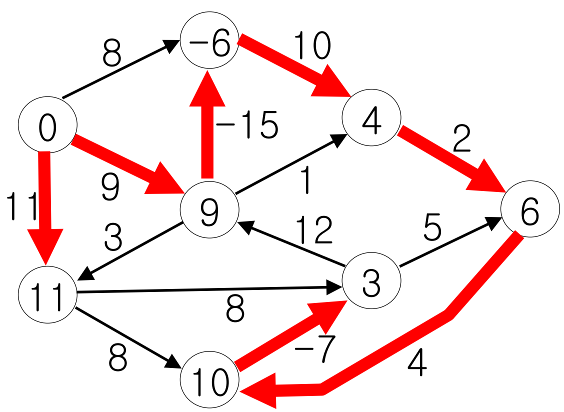
0
−6
4
9
6
11
3
10
8
10
−15
2
9
1
11
3
12
5
8
8
−7
4

(h) $i = 7$



0
−6
4
9
6
11
3
10
8
10
−15
2
9
1
11
3
12
5
8
8
−7
4

(i)



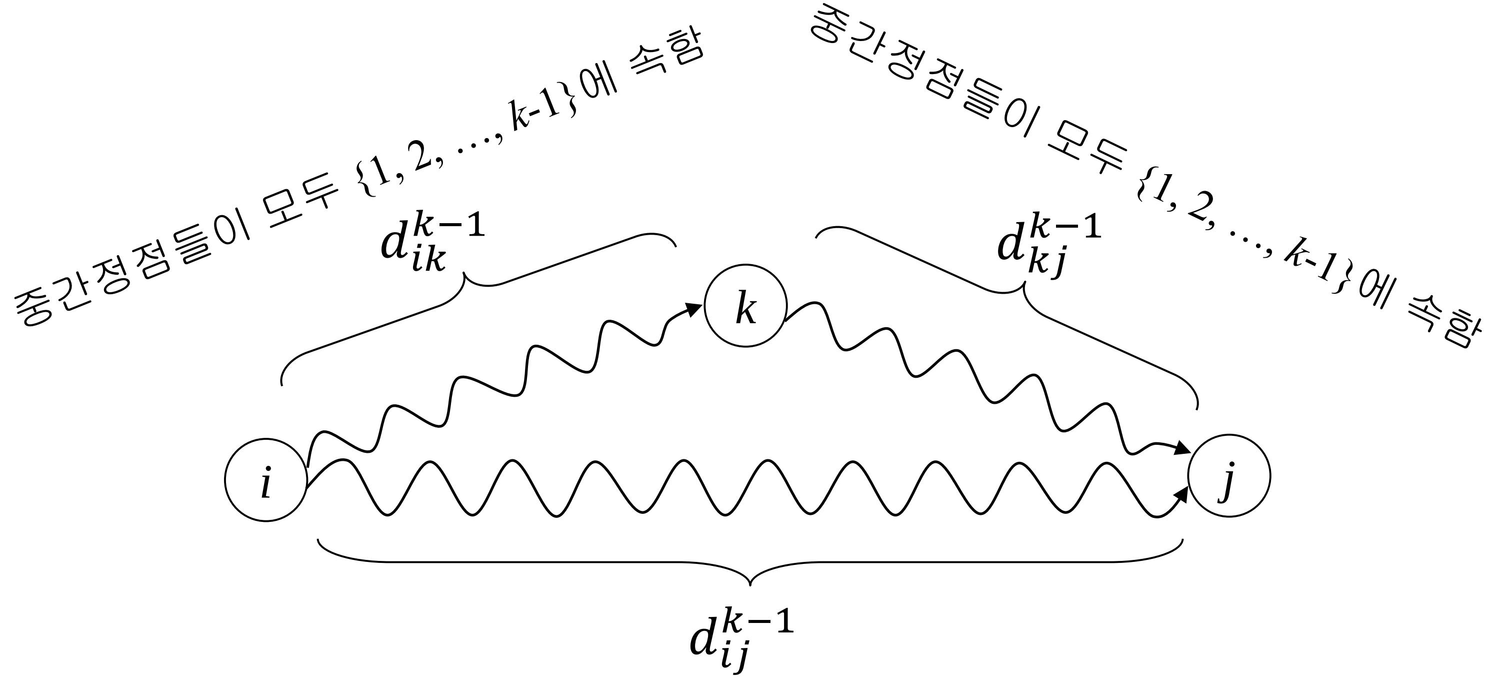
0
−6
4
9
6
11
3
10
8
10
−15
2
9
1
11
3
12
5
8
8
−7
4

플로이드-워샬Floyd-Warshall 알고리즘

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

$d_{ij}^{\,k}$: vertex set $\{v_1, v_2, \ldots, v_k\}$에 속하는 것들만 거쳐 v_i에서 v_j에 이르는 최단경로 길이

$$d_{ij}^{\,k} = \begin{cases} w_{ij}\,, & k = 0 \\ \min\{d_{ij}^{\,k\text{-}1},\ d_{ik}^{\,k\text{-}1} + d_{kj}^{\,k\text{-}1}\}, & k \geq 1 \end{cases}$$



중간정점들이 모두 $\{1, 2, \ldots, k\text{-}1\}$에 속함

플로이드-워샬 알고리즘

```
FloydWarshall(G)
{
    for i ← 1 to n
        for j ← 1 to n
            d^0_ij ← w_ij ;
    for k ← 1 to n                      ▷ 중간정점 집합 {1, 2, …, k}
        for i ← 1 to n                  ▷ i : 시작 정점
            for j ← 1 to n              ▷ j : 마지막 정점
                d^k_ij ← min {d^(k-1)_ij , d^(k-1)_ik + d^(k-1)_kj};
}
```

✓수행시간: $\Theta(|V|^3)$

✓문제의 총 수 $\Theta(|V|^3)$, 각 문제의 계산에 $\Theta(1)$

싸이클이 없는 그래프의 최단경로

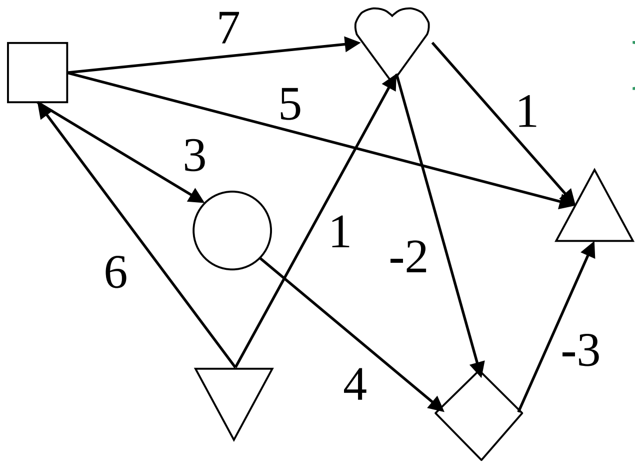
- 싸이클이 없는 유향 그래프를 DAG라 한다
  - DAG: Directed Acyclic Graph
- DAG에서의 최단경로는 선형시간에 간단히 구할 수 있다

```
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} ;
}
```

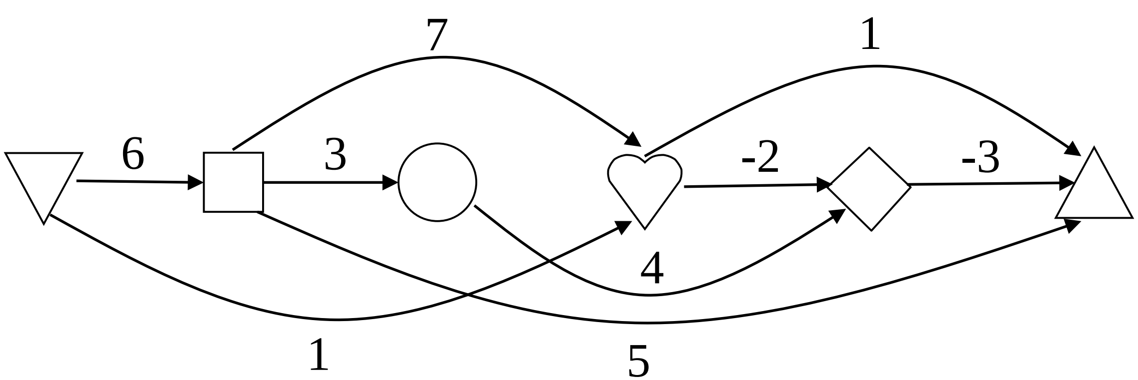
✓수행시간: $\Theta(|V|+|E|)$

DAG-ShortestPath의 작동 예

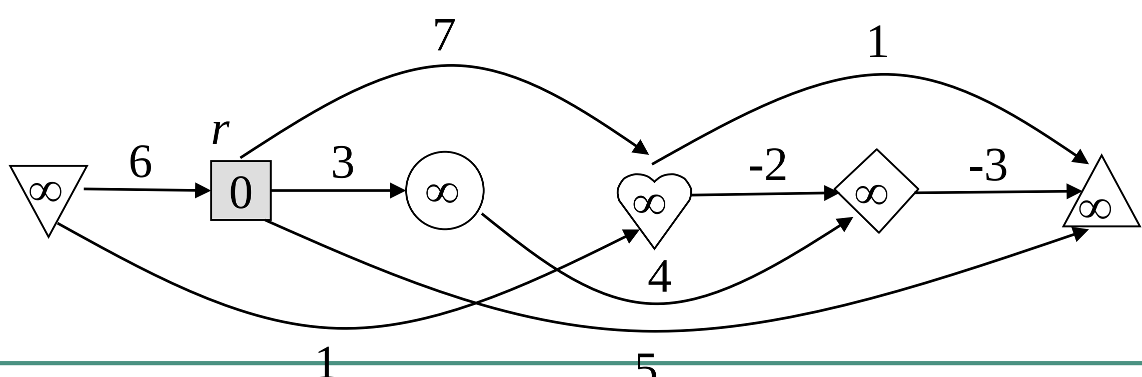
(a)



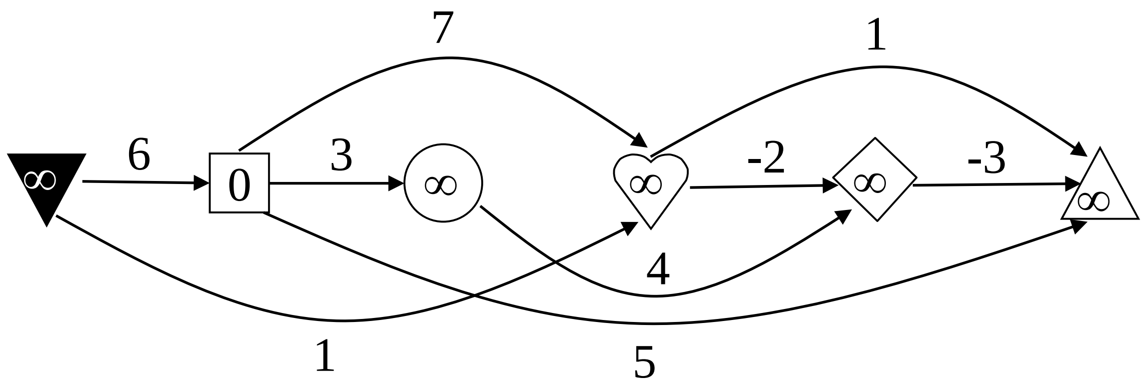
(b)



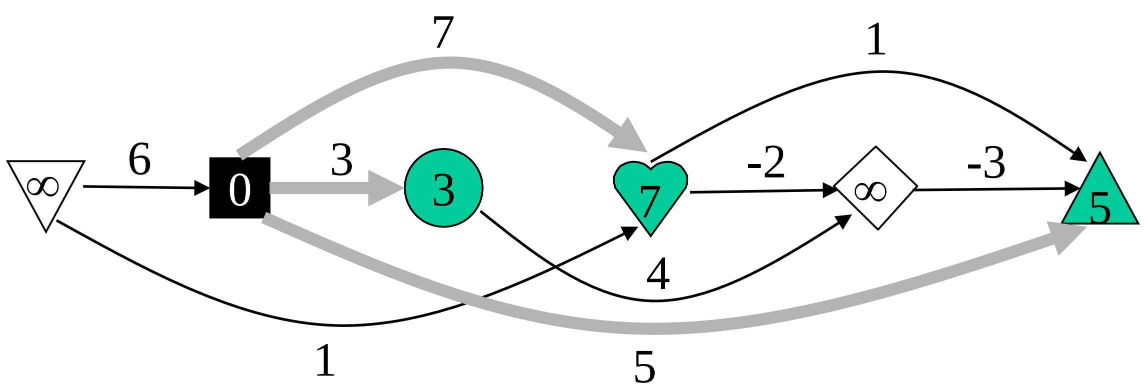
(c)



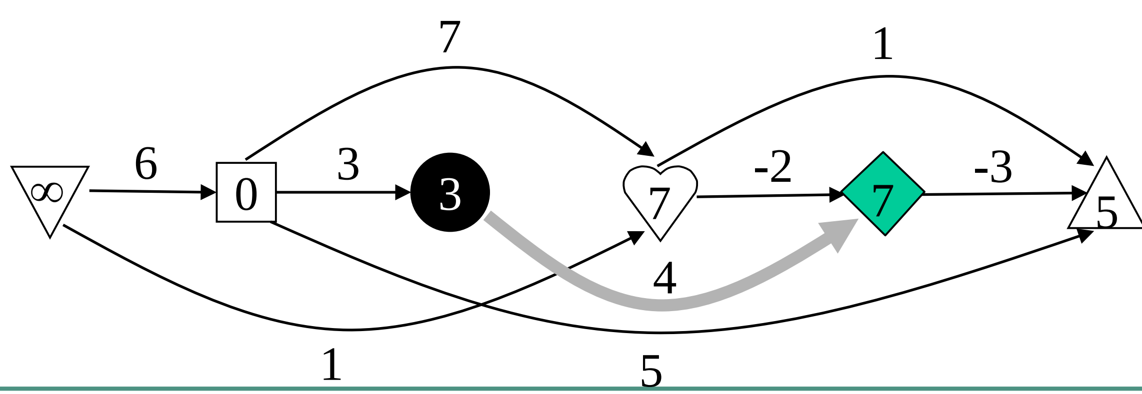
(d)



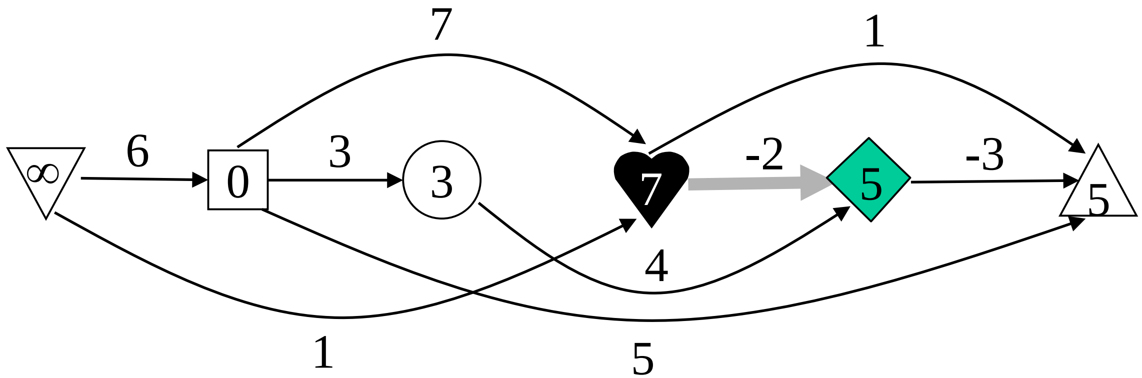
(e)



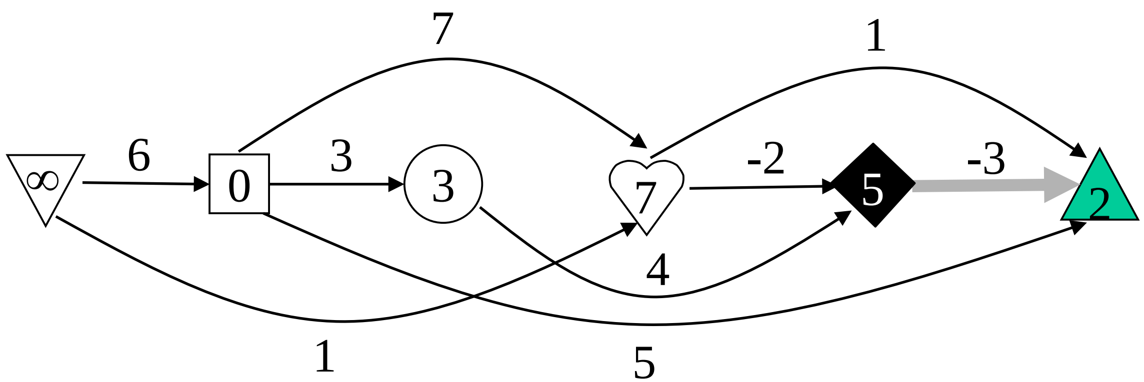
(f)



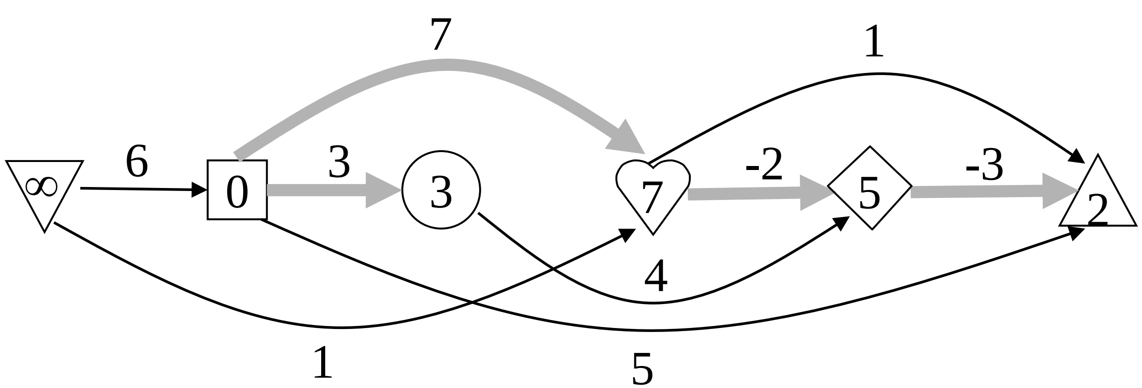
(g)



(h)



(i)



강연결요소 구하기

- 강하게 연결됨
  - 유향 그래프의 모든 정점쌍에 대해서 양방향으로 경로가 존재하면 강하게 연결되었다고 한다
  - 강하게 연결된 부분 그래프를 강연결요소Strongly connected component라 한다
- 임의의 그래프에서 강연결요소들을 찾는 알고리즘을 공부한다

강연결요소 구하기 알고리즘

stronglyConnectedComponent(G)
{

1 그래프 G에 대해 DFS를 수행하여 각 정점 v의 완료시간 $f[v]$를 계산한다.

2 G의 모든 간선들의 방향을 뒤집어 G^R을 만든다.

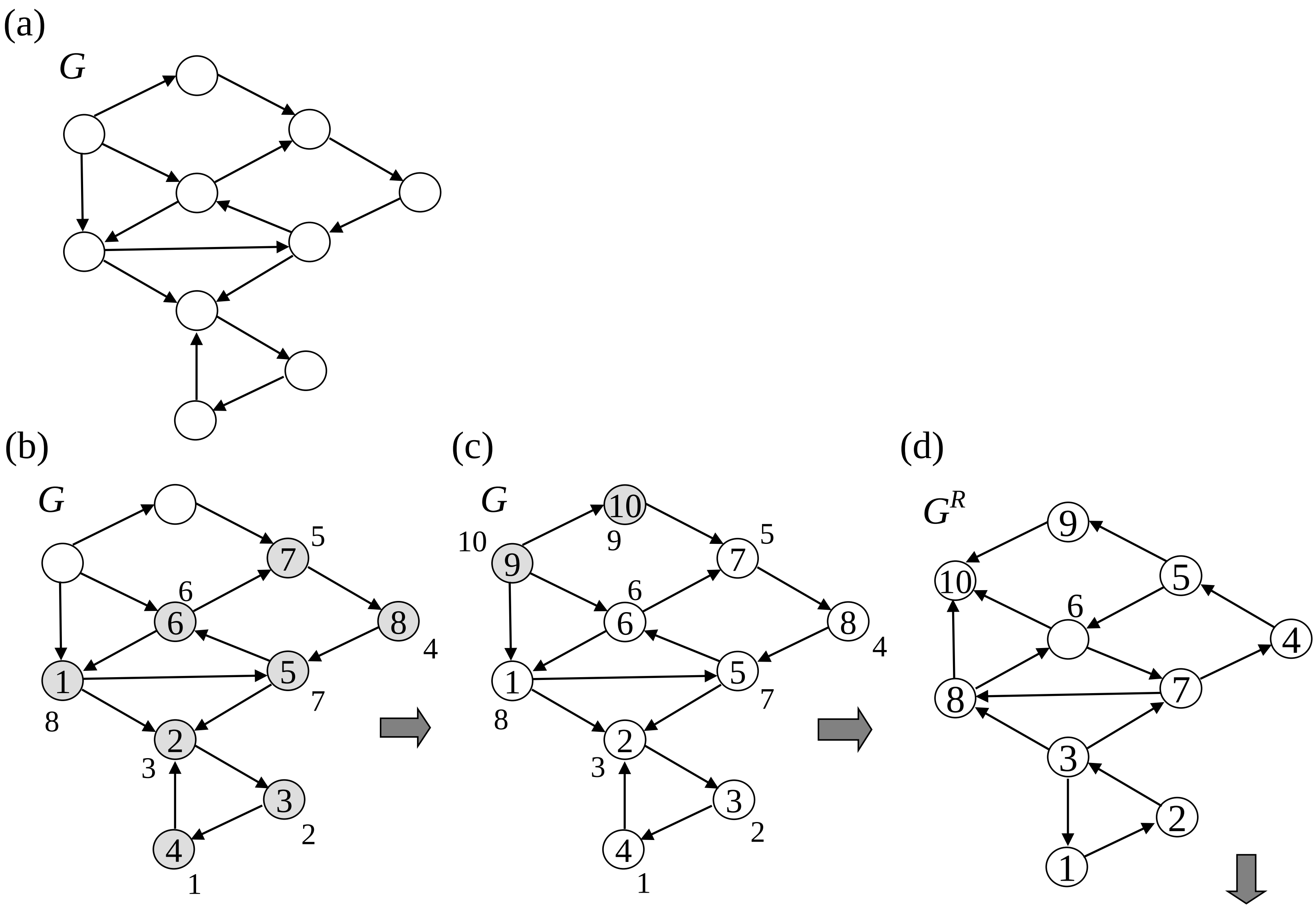
3 DFS(G^R)를 수행하되 [알고리즘 10-2]의 ❶행에서 시작점을 택할 때 1에서 구한 $f[v]$가 가장 큰 정점으로 잡는다.

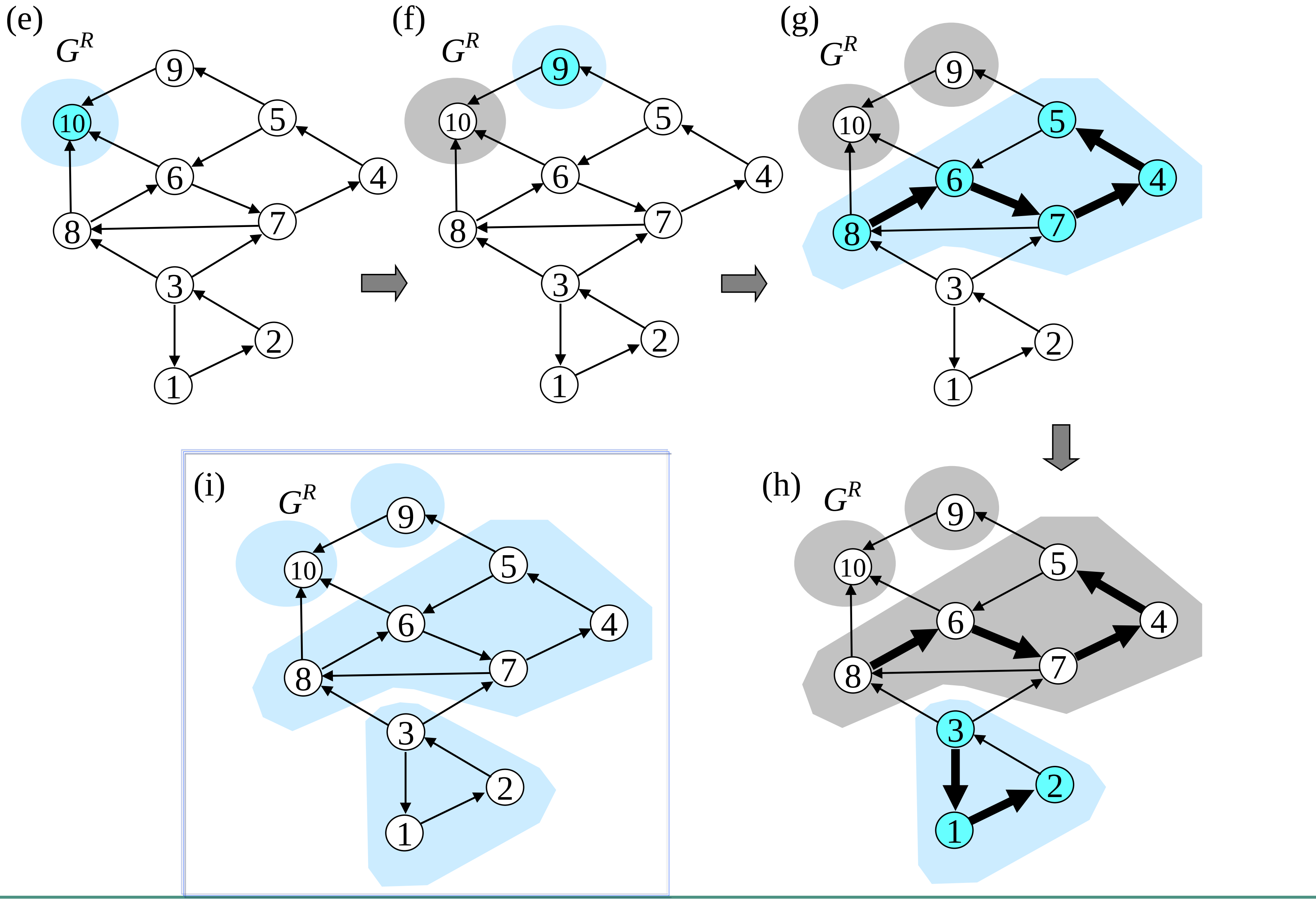
4 앞의 3에서 만들어진 분리된 트리들 각각을 강연결요소로 리턴한다.

}

✓수행 시간: $\Theta(|V|+|E|)$

stronglyConnectedComponent의 작동 예




(e)
G^R
(f)
G^R
(g)
G^R
(h)
G^R
(i)
G^R

Thank you