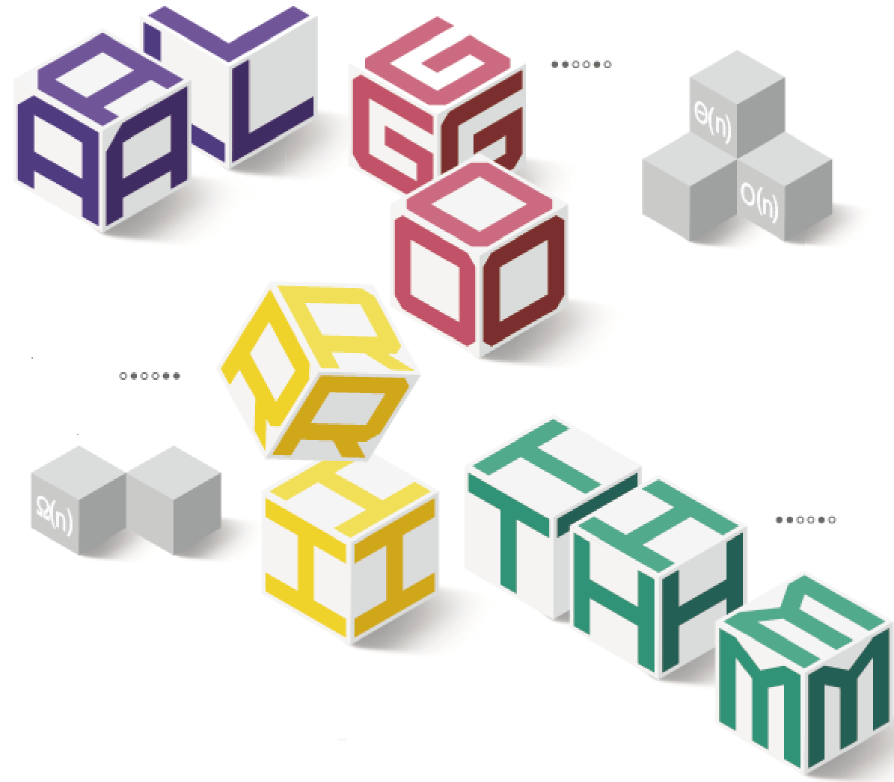


관계 중심의 사고법

쉽게 배우는 알고리즘

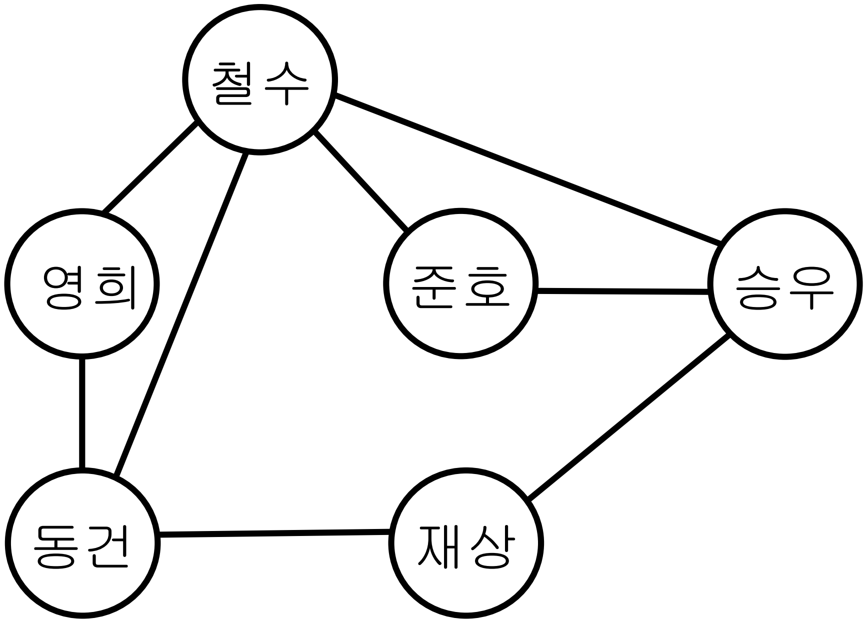
10장. 그래프 알고리즘

Graph Algorithms

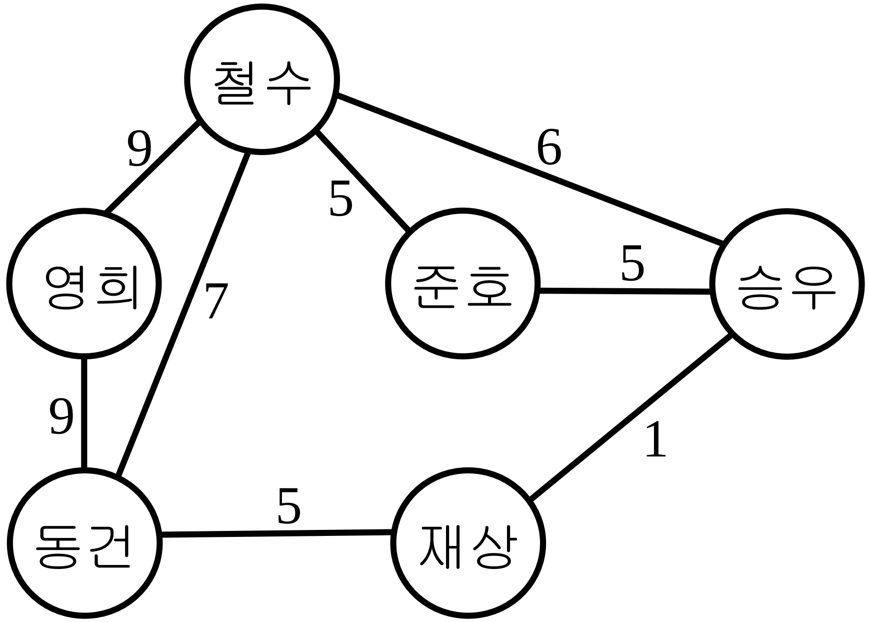
Graph

- 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

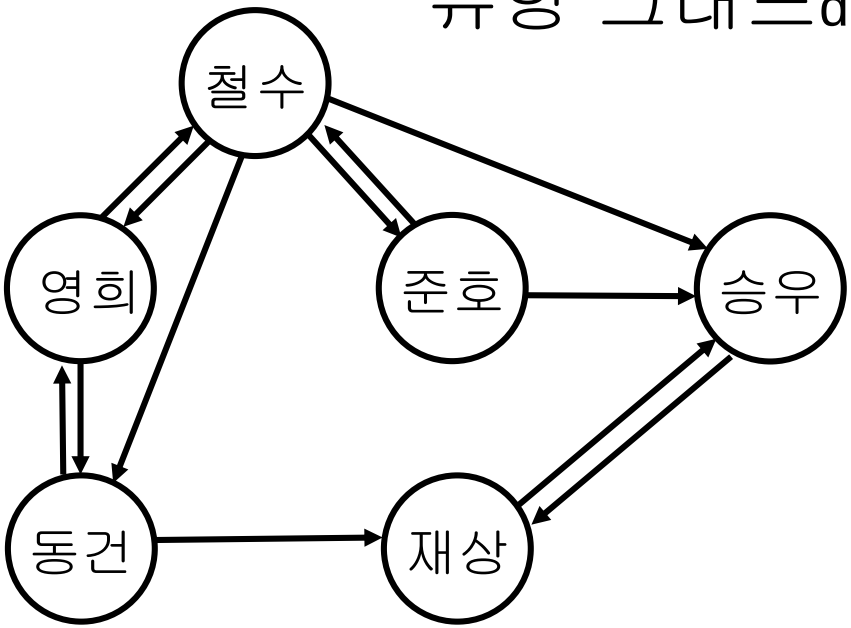


사람들간의 친분 관계를 나타낸 그래프

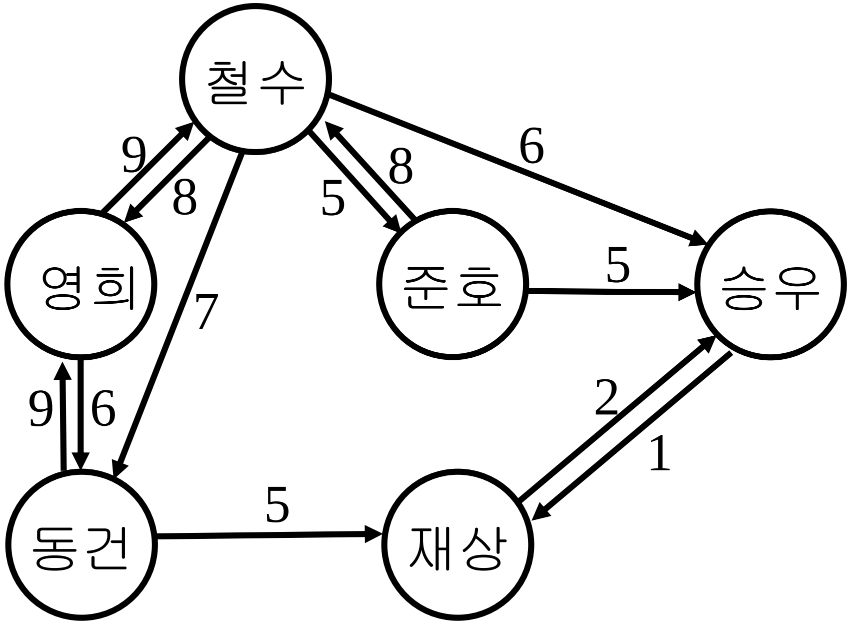


친밀도를 가중치로 나타낸 친분관계 그래프

유향 그래프directed graph=digraph



방향을 고려한 친분관계 그래프

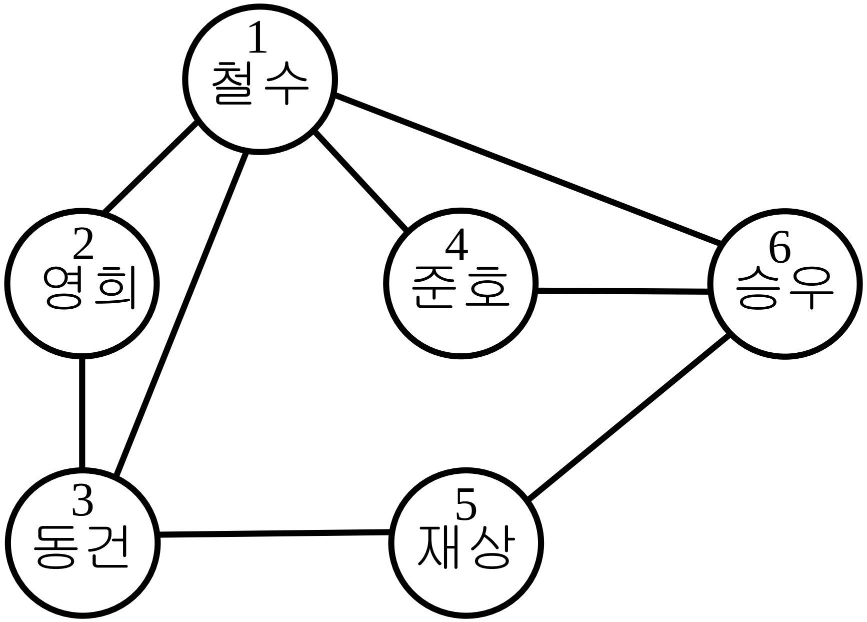


가중치를 가진 유향 그래프

Graph의 표현 1: Adjacency Matrix

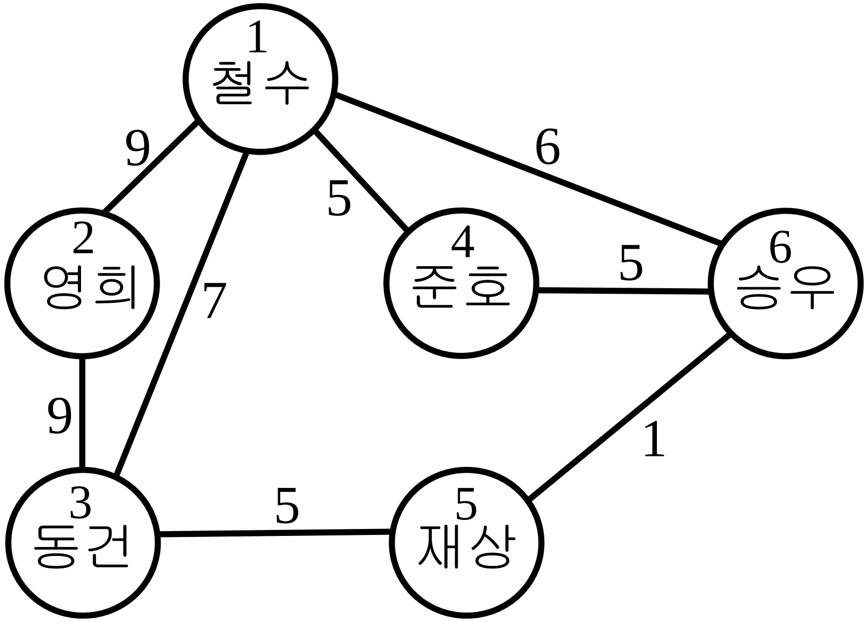
N: 정점의 총 수

- Adjacency matrix
  - NxN 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



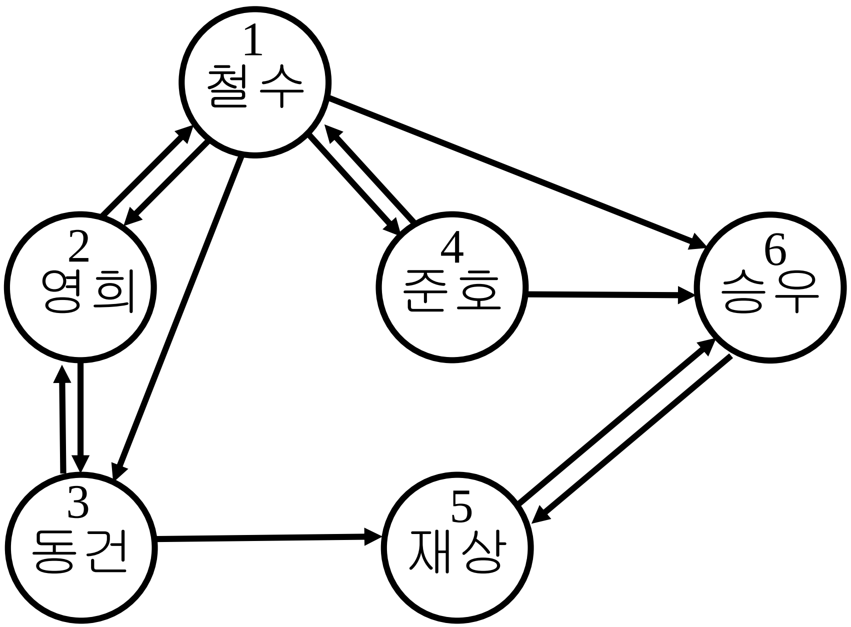
	1	2	3	4	5	6
1	0	1	1	1	0	1
2	1	0	1	0	0	0
3	1	1	0	0	1	0
4	1	0	0	0	0	1
5	0	0	1	0	0	1
6	1	0	0	1	1	0

무향 그래프의 예



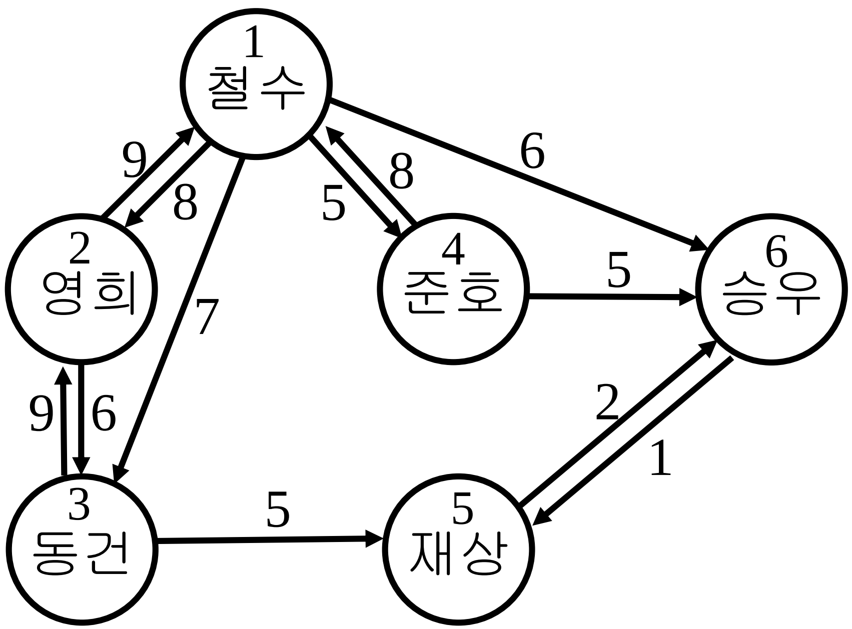
	1	2	3	4	5	6
1	0	9	7	5	0	6
2	9	0	9	0	0	0
3	7	9	0	0	5	0
4	5	0	0	0	0	5
5	0	0	5	0	0	1
6	6	0	0	5	1	0

가중치 있는 무향 그래프의 예



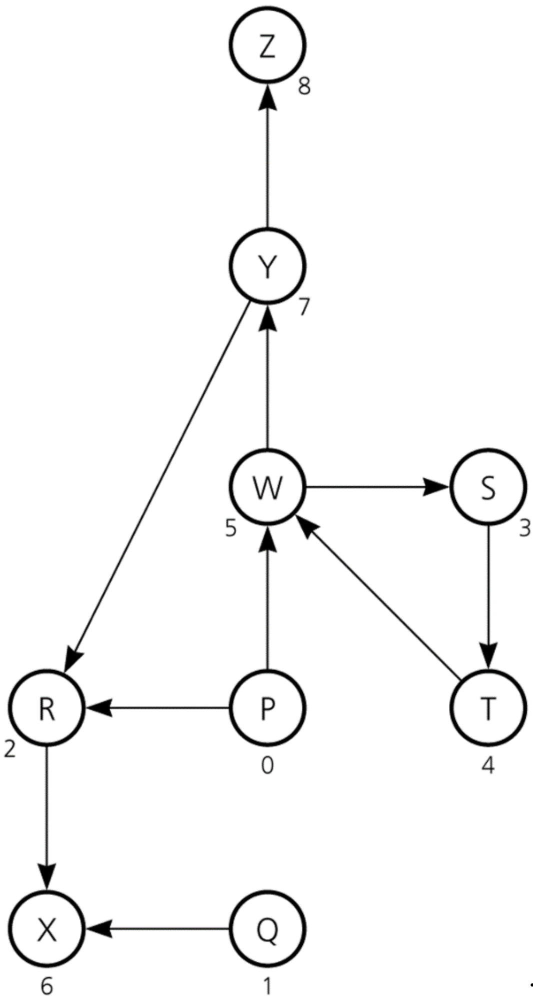
	1	2	3	4	5	6
1	0	1	1	1	0	1
2	1	0	1	0	0	0
3	0	1	0	0	1	0
4	1	0	0	0	0	1
5	0	0	0	0	0	1
6	0	0	0	0	1	0

유향 그래프의 예



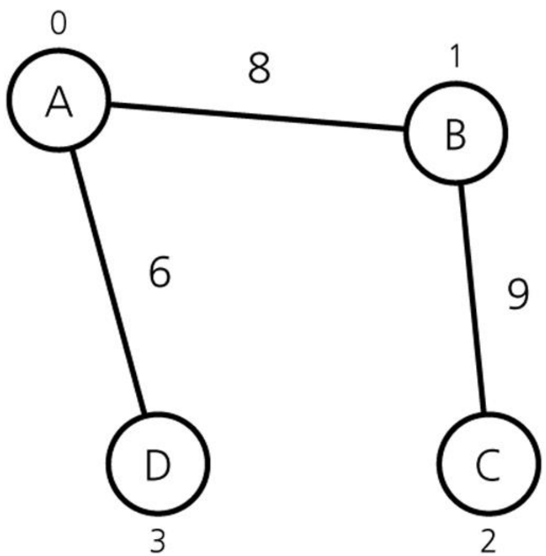
	1	2	3	4	5	6
1	0	8	7	5	0	6
2	9	0	6	0	0	0
3	0	9	0	0	5	0
4	8	0	0	0	0	5
5	0	0	0	0	0	2
6	0	0	0	0	1	0

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	∞	∞	∞

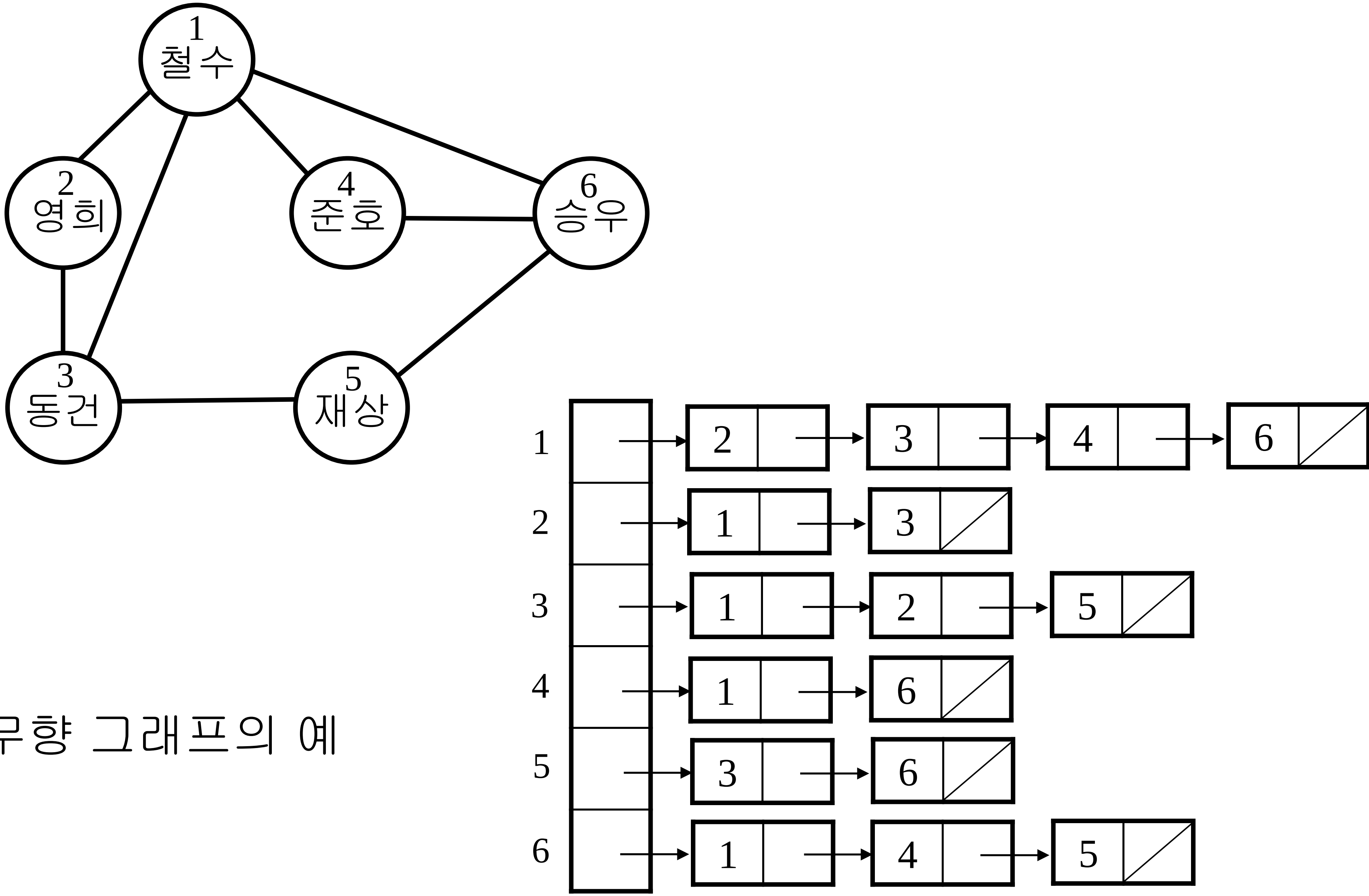
가중치 있는 그래프의 다른 예

관련 자원

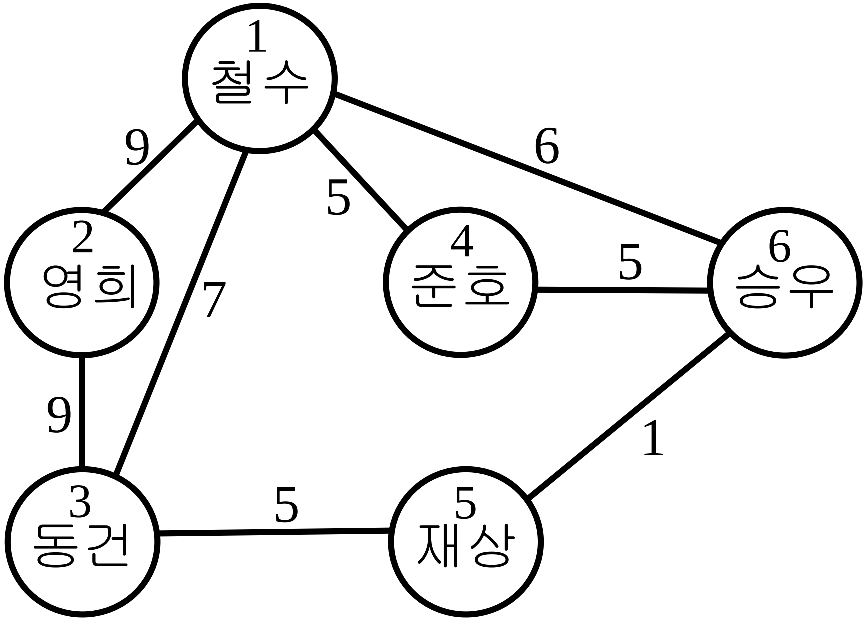
- 표현을 위한 공간 크기?
- 두 vertex의 인접성 체크?
- 한 vertex에 인접한 모든 vertex 확인?

Graph의 표현 2: Adjacency List

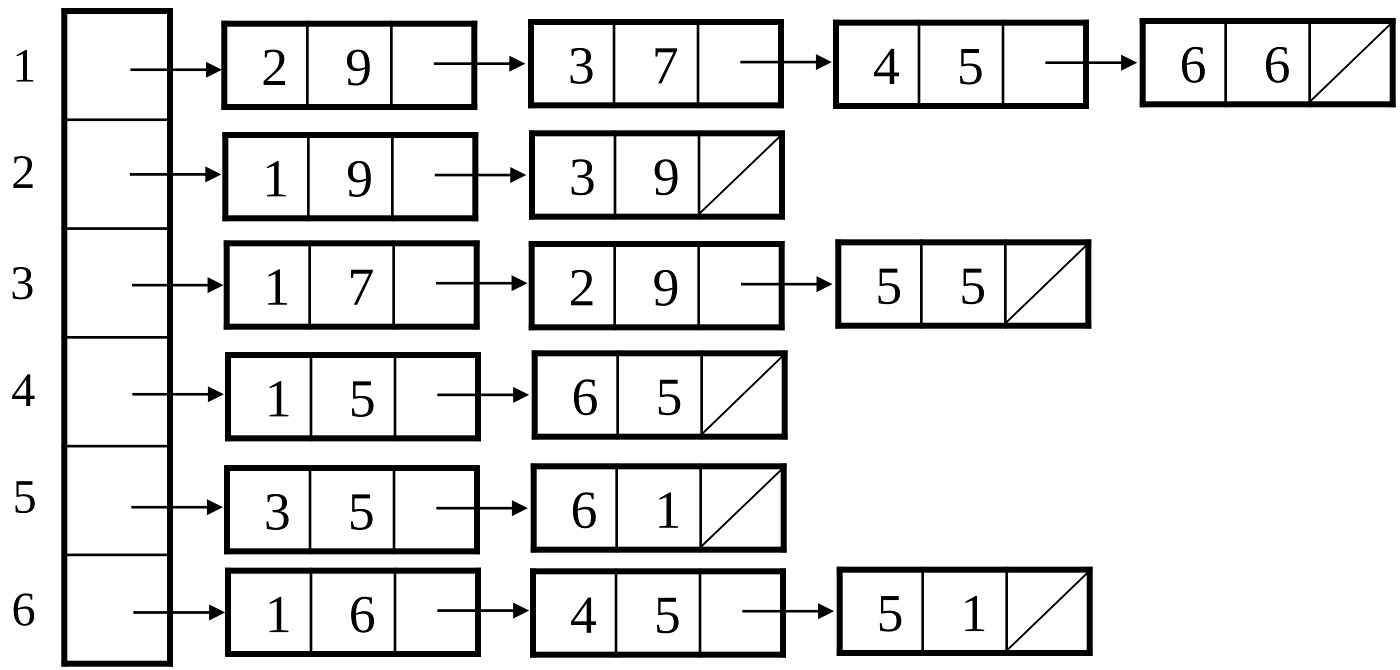
- Adjacency list
  - N개의 연결 리스트로 표현
  - i번째 리스트는 정점 i에 인접한 정점들을 리스트로 연결해 놓음
  - 가중치 있는 그래프의 경우
    - 리스트는 가중치도 보관한다



무향 그래프의 예


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

가중치 있는 그래프의 예

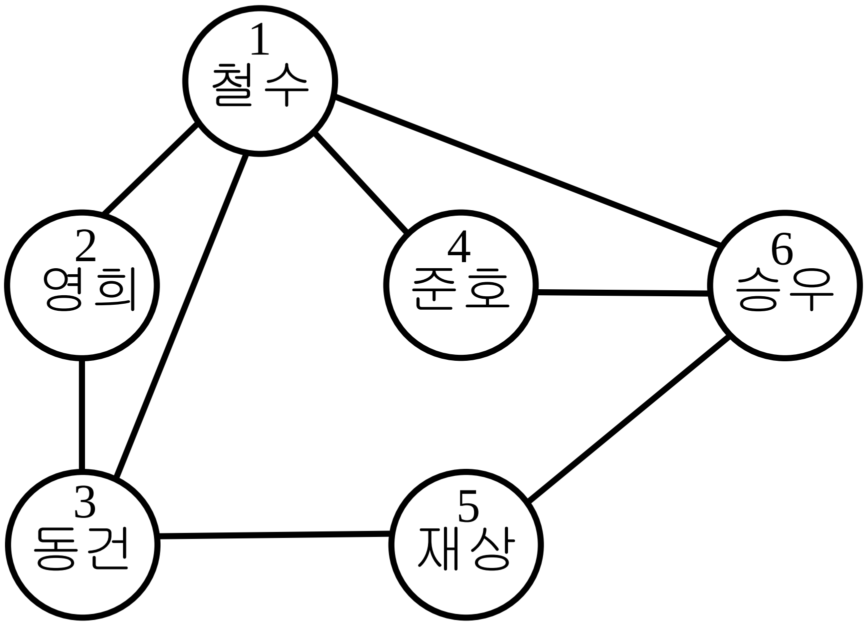

1
2 9
3 7
4 5
6 6
2
1 9
3 9
3
1 7
2 9
5 5
4
1 5
6 5
5
3 5
6 1
6
1 6
4 5
5 1

관련 자원

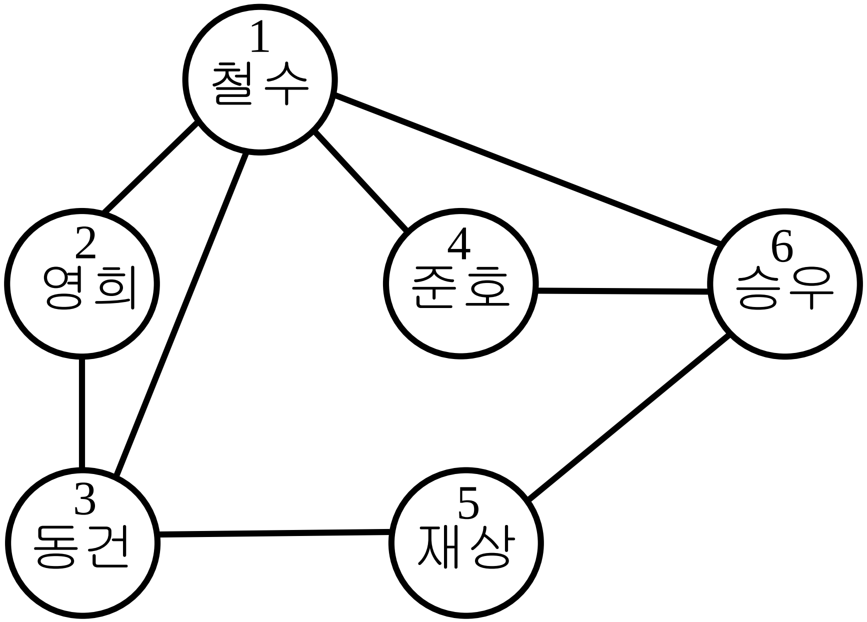
- 표현을 위한 공간 크기?
- 두 vertex의 인접성 체크?
- 한 vertex에 인접한 모든 vertex 확인?

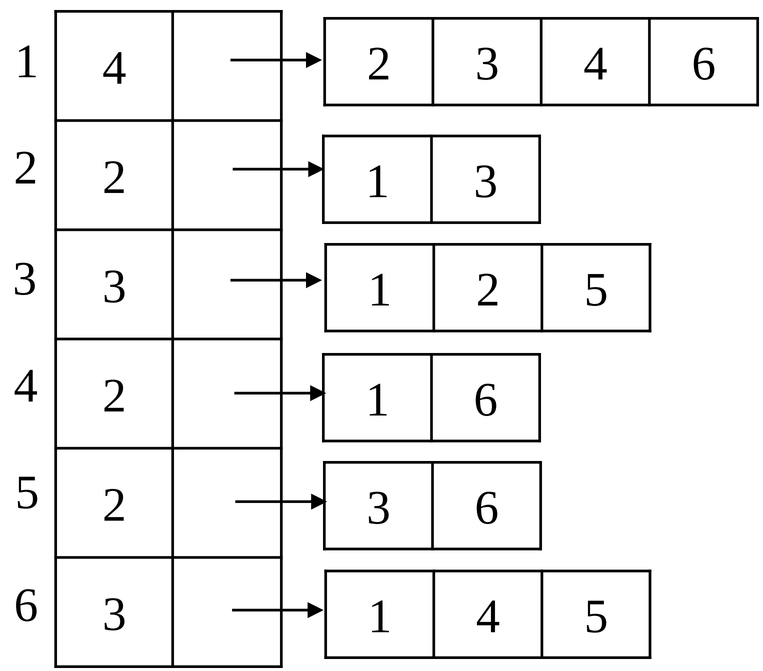
Graph의 표현 3: Adjacency Array (Adjacency List의 변형)

- Adjacency array
  - N개의 배열로 표현
  - i번째 배열은 정점 i에 인접한 정점들을 배열에 정렬된 형태로 저장해 놓음
  - 가중치 있는 그래프의 경우
    - 배열은 가중치도 보관한다

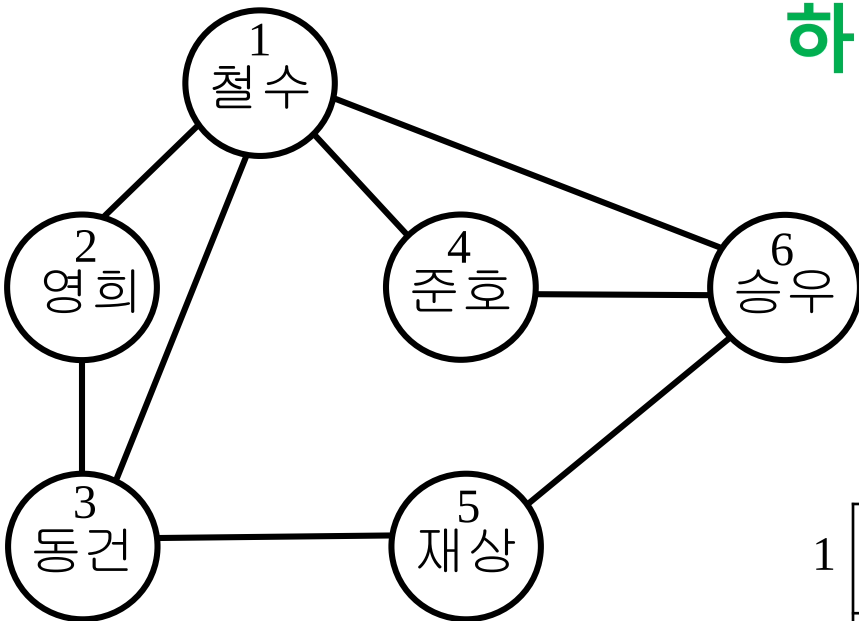


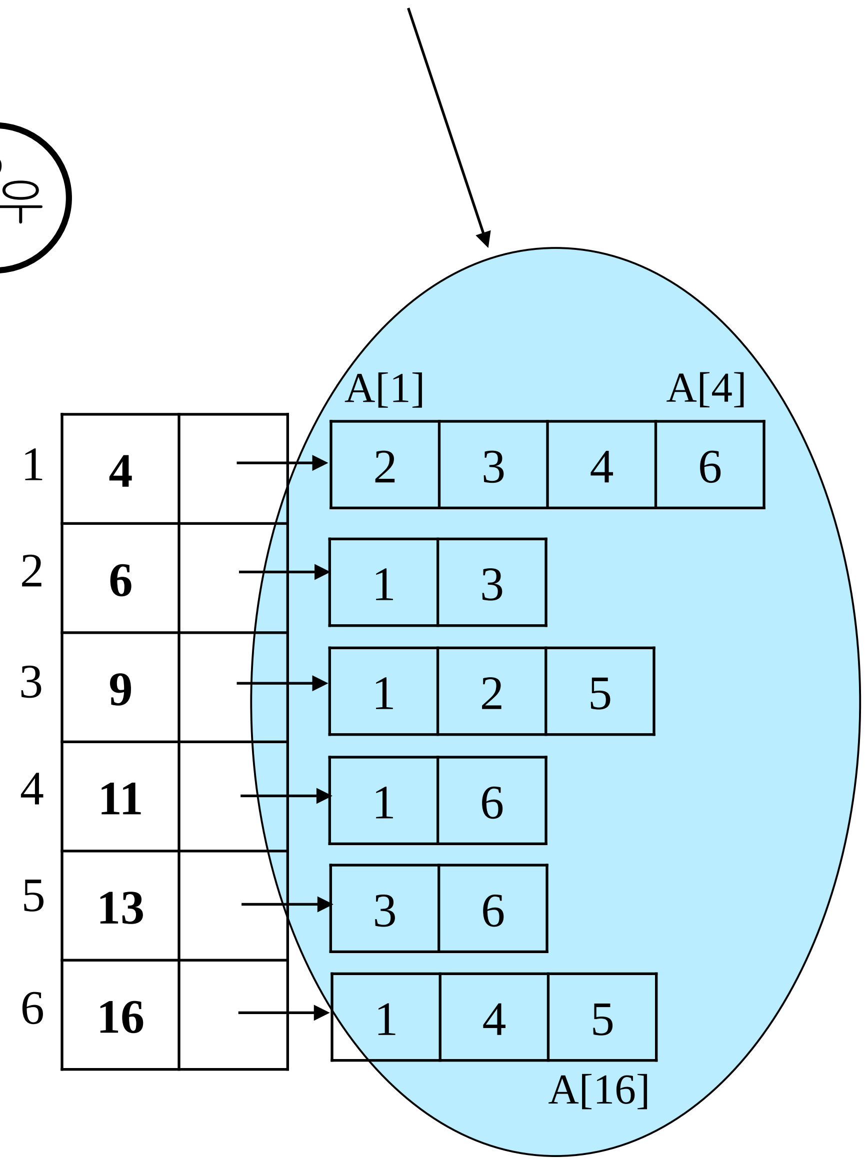
무향 그래프의 예


1
철수
2
영희
4
준호
6
승우
3
동건
5
재상


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

하나의 배열을 쓸 수도 있다



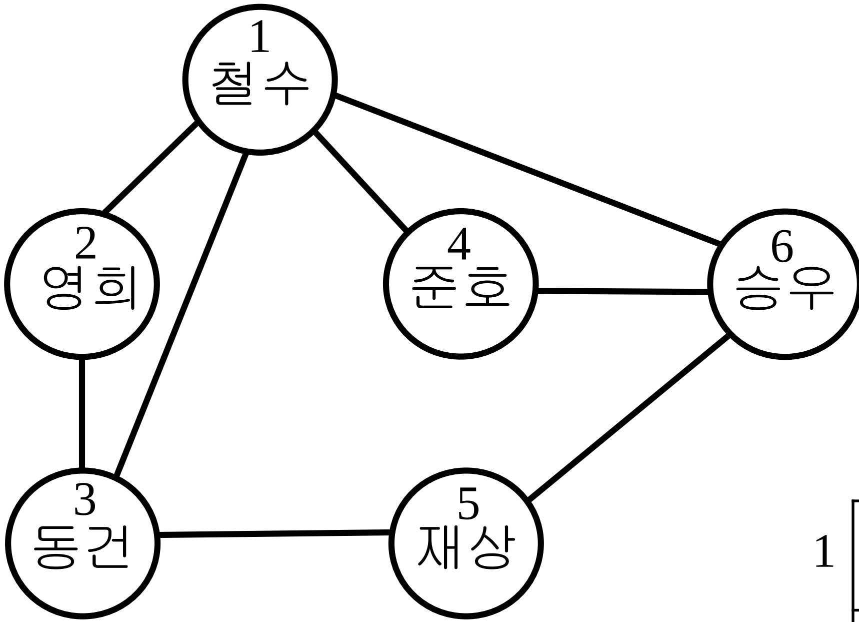


관련 자원

- 표현을 위한 공간 크기?
- 두 vertex의 인접성 체크?
- 한 vertex에 인접한 모든 vertex 확인?

Graph의 표현 4: Adjacency Hash Table

- Adjacency hash table
  - N 개의 hash table로 표현
  - i 번째 hash table은 정점 i 에 인접한 정점들을 hash table에 저장해 놓음
  - 가중치 있는 그래프의 경우
    - Hash table은 가중치도 보관한다



N개의 hash table

1	**4**	{2, 3, 4, 6}
2	**6**	{1, 3}
3	**9**	{1, 2, 5}
4	**11**	{1, 6}
5	**13**	{3, 6}
6	**16**	{1, 4, 5}

관련 자원

- 표현을 위한 공간 크기?
- 두 vertex의 인접성 체크?
- 한 vertex에 인접한 모든 vertex 확인?

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

DFS깊이우선탐색

DFS (v)
{
    v.visited ← YES;
    for each $x \in v$.adjList   ▷ v.adjList : 정점 v에 연결된 정점 집합
        if (x.visited = NO) **then** DFS(x);
}

✓수행시간: $\Theta(V+E)$

편의상 $|V|$, $|E|$ 대신 V, E로 표기

BFS너비우선탐색

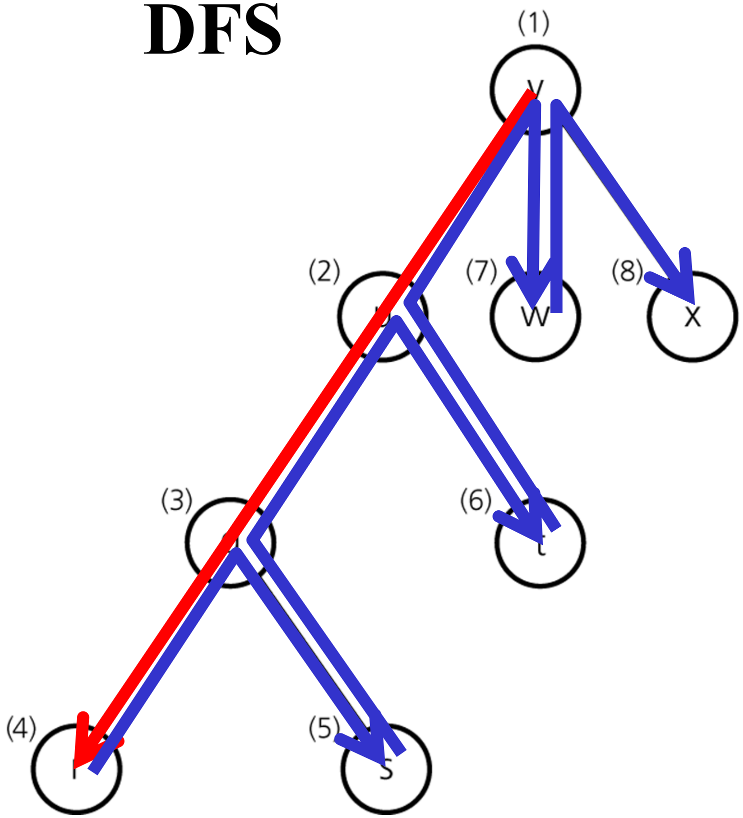
```
BFS(G, s)
{
        for each v ∈ V
                v.visited ← NO;
        s.visited ← YES;
        enqueue(Q, s);
        while (Q ≠Φ) {
                x ← dequeue(Q);
                for each v ∈ x.adjList
                        if (v.visited = NO) then {
                                v.visited ← YES;
                                enqueue(Q, v);
                        }

        }
}
```

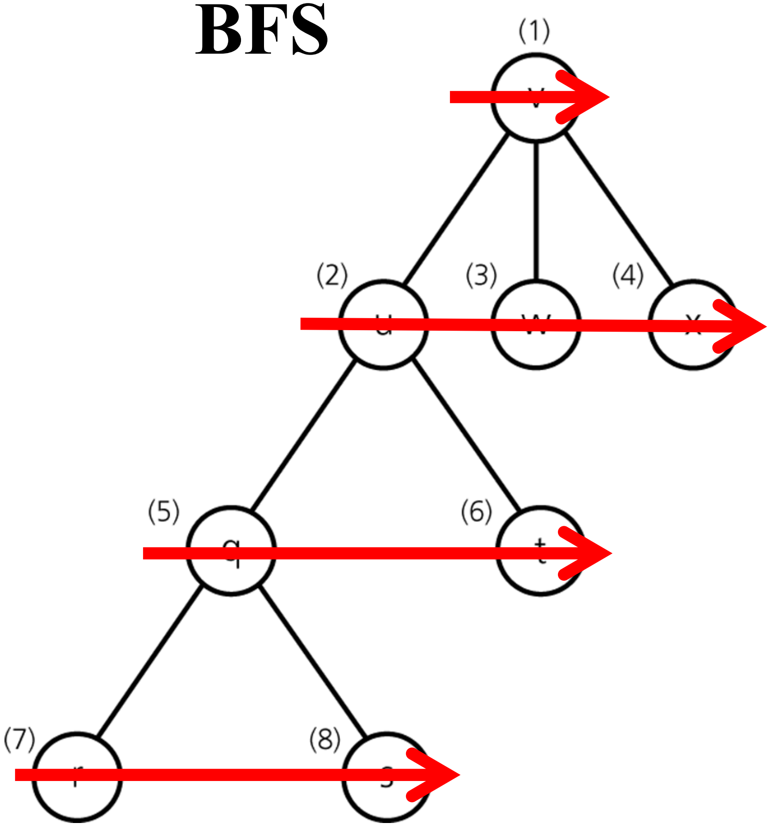
✓수행시간: $\Theta(V+E)$

동일한 Tree를 각각 DFS/BFS로 방문하기

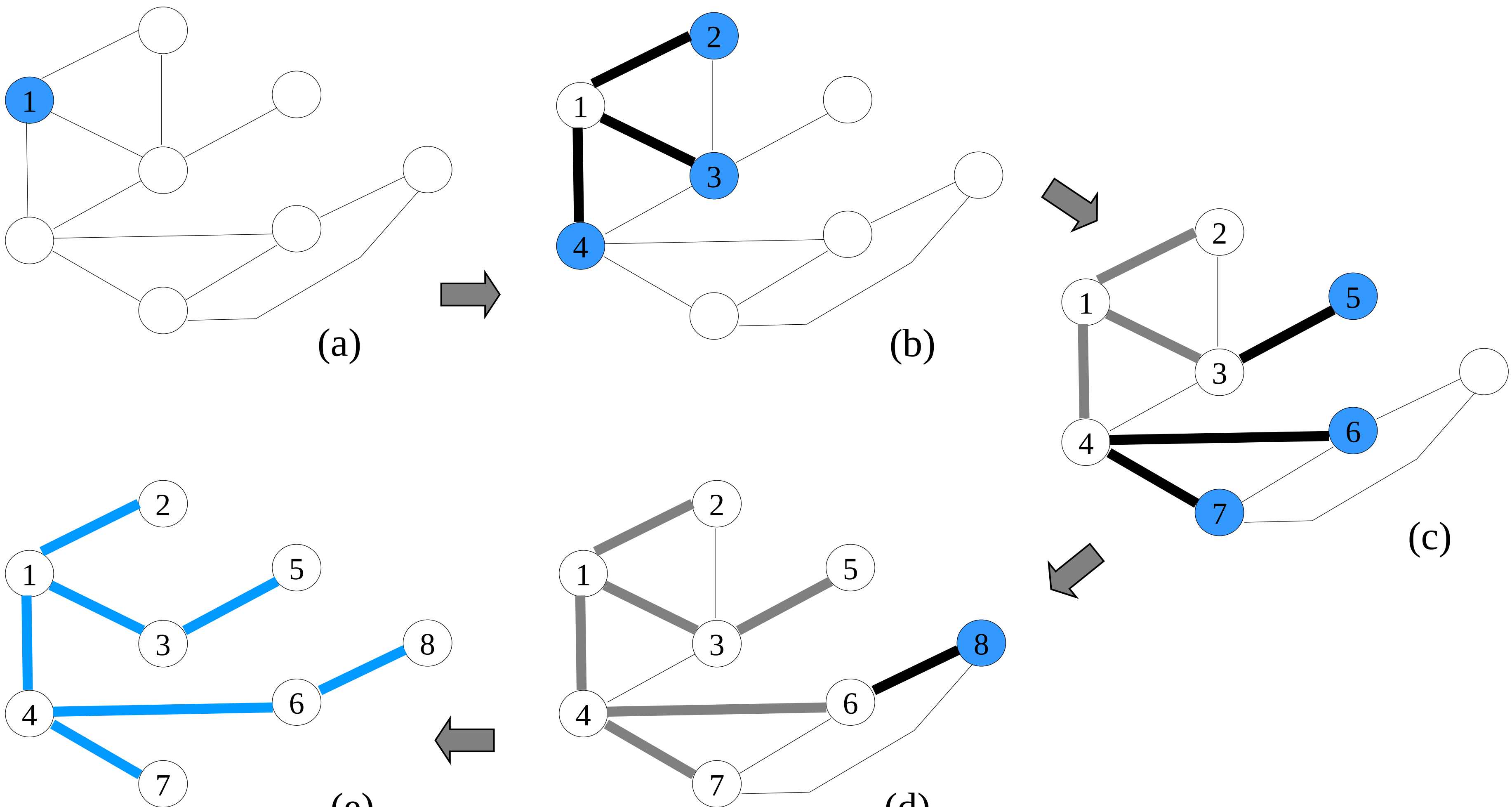
DFS



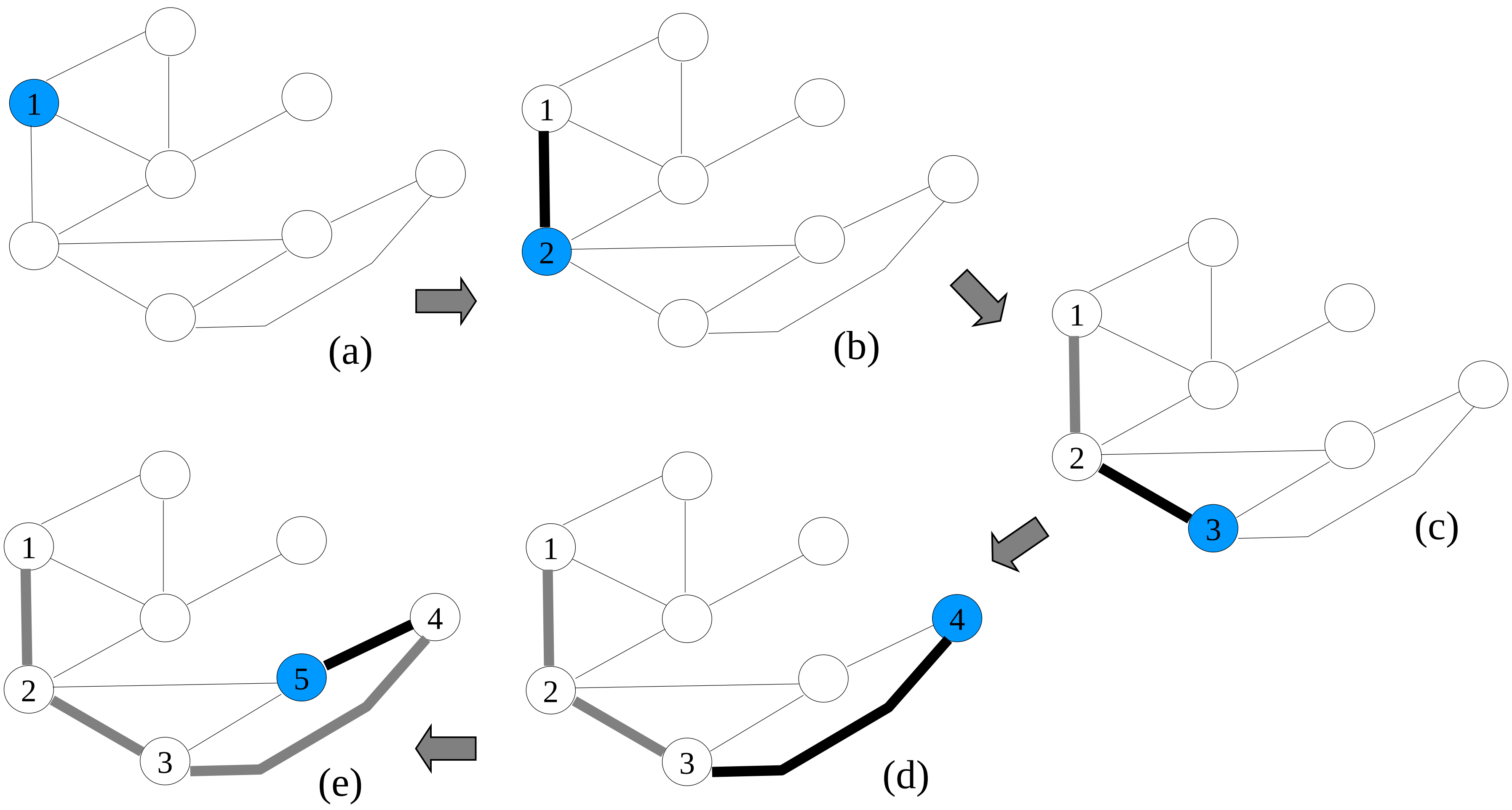
BFS



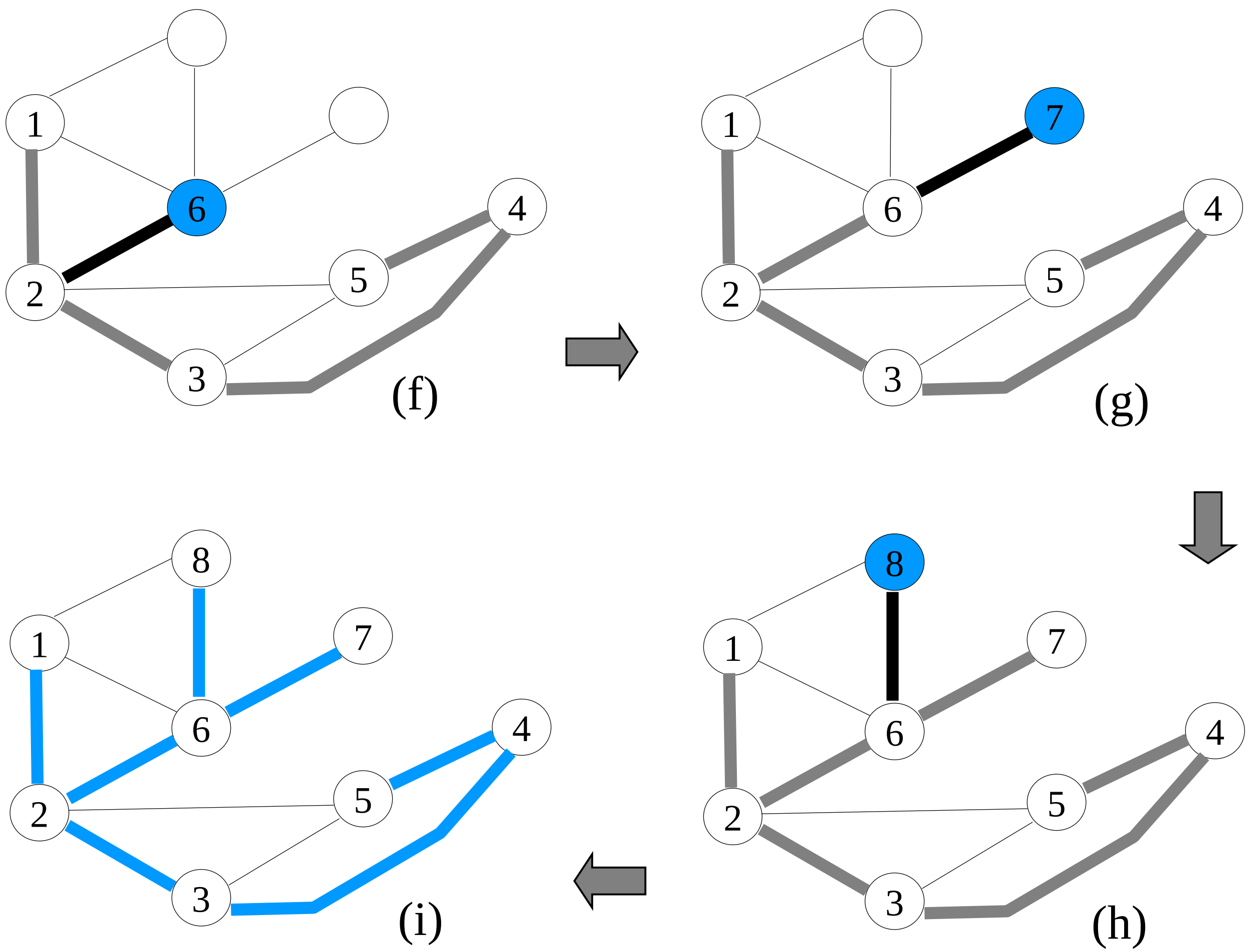
BFS의 작동 예



DFS의 작동 예



DFS의 작동 예 (계속)



Classification of Edges

- Tree edge
  - BFS/DFS에서 vertex를 tree에 편입시킬 때 사용한 edge
- 나머지는…
  - Back edge
    - BFS/DFS tree에서 자신의 ancestor로 연결되는 edge
  - Forward edge
    - BFS/DFS tree에서 자신의 descendant로 연결되는 edge
  - Cross edge
    - Ancestor/descendant 관계가 아닌 두 vertex를 연결하는 edge

Undirected vs. Directed

O: 존재 가능, X: 존재 불가능

Undirected Graph	Forward edge = back edge	Cross edge
BFS tree	X	O
DFS tree	O	X

Directed Graph	Forward edge	Back edge	Cross edge
BFS tree	X	O	O
DFS tree	O	O	O

✓ 이 관점으로 따져보는 것은 BFS와 DFS에 대한 이해를 깊게 해준다

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

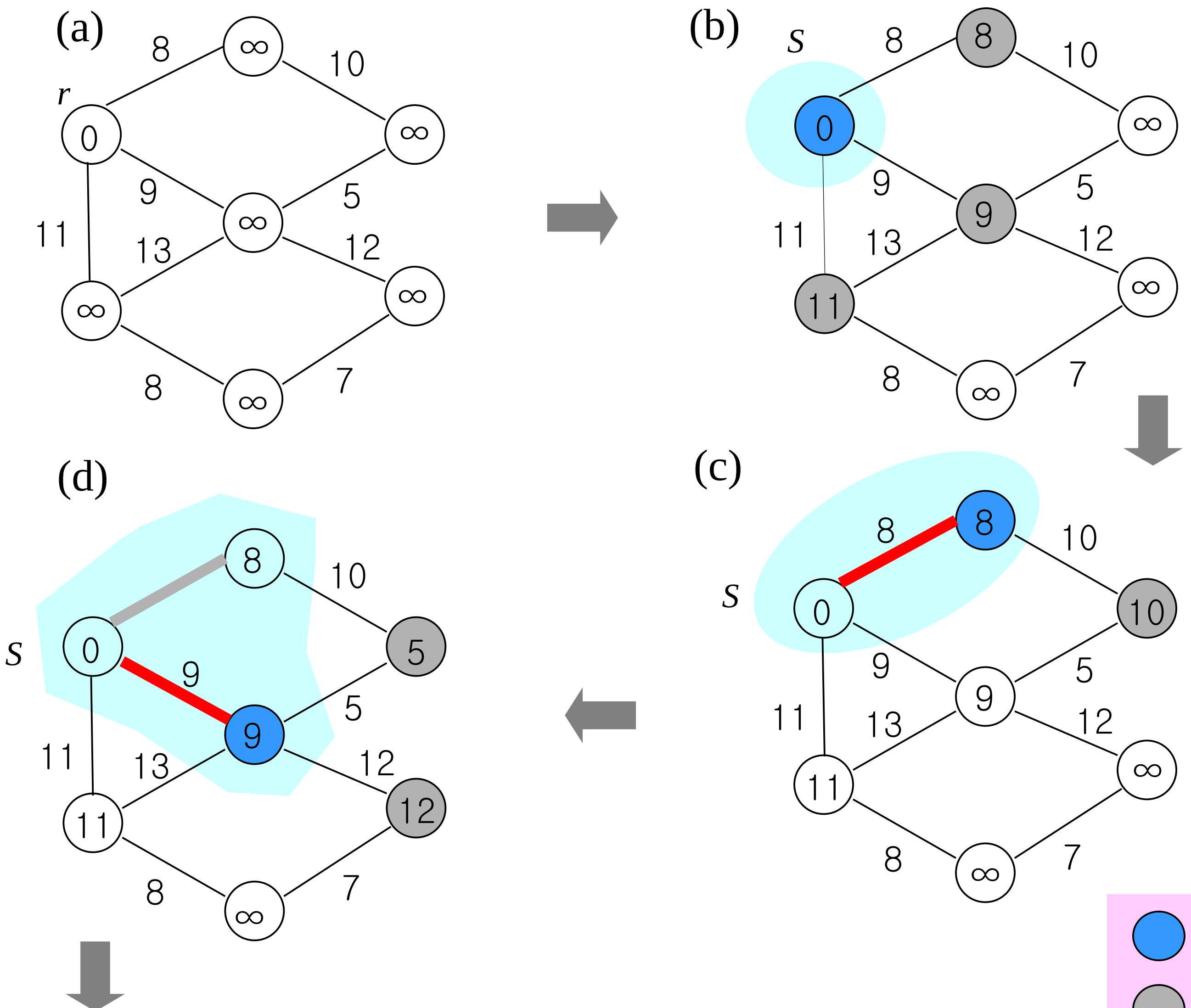
Prim (G, r)
{
    $S \leftarrow \Phi$;
    정점 r을 방문되었다고 표시하고, 집합 S에 포함시킨다;
    while ($S \neq V$) {
        S에서 V-S를 연결하는 간선들 중 최소길이의 간선 (x, y) 를 찾는다; ▷ ($x \in S$, $y \in V$-S)
        정점 y를 방문되었다고 표시하고, 집합 S에 포함시킨다;
        (정점 y로 relaxation 할 수 있으면 한다)
    }
}

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

✓수행시간: $O(E\log V)$

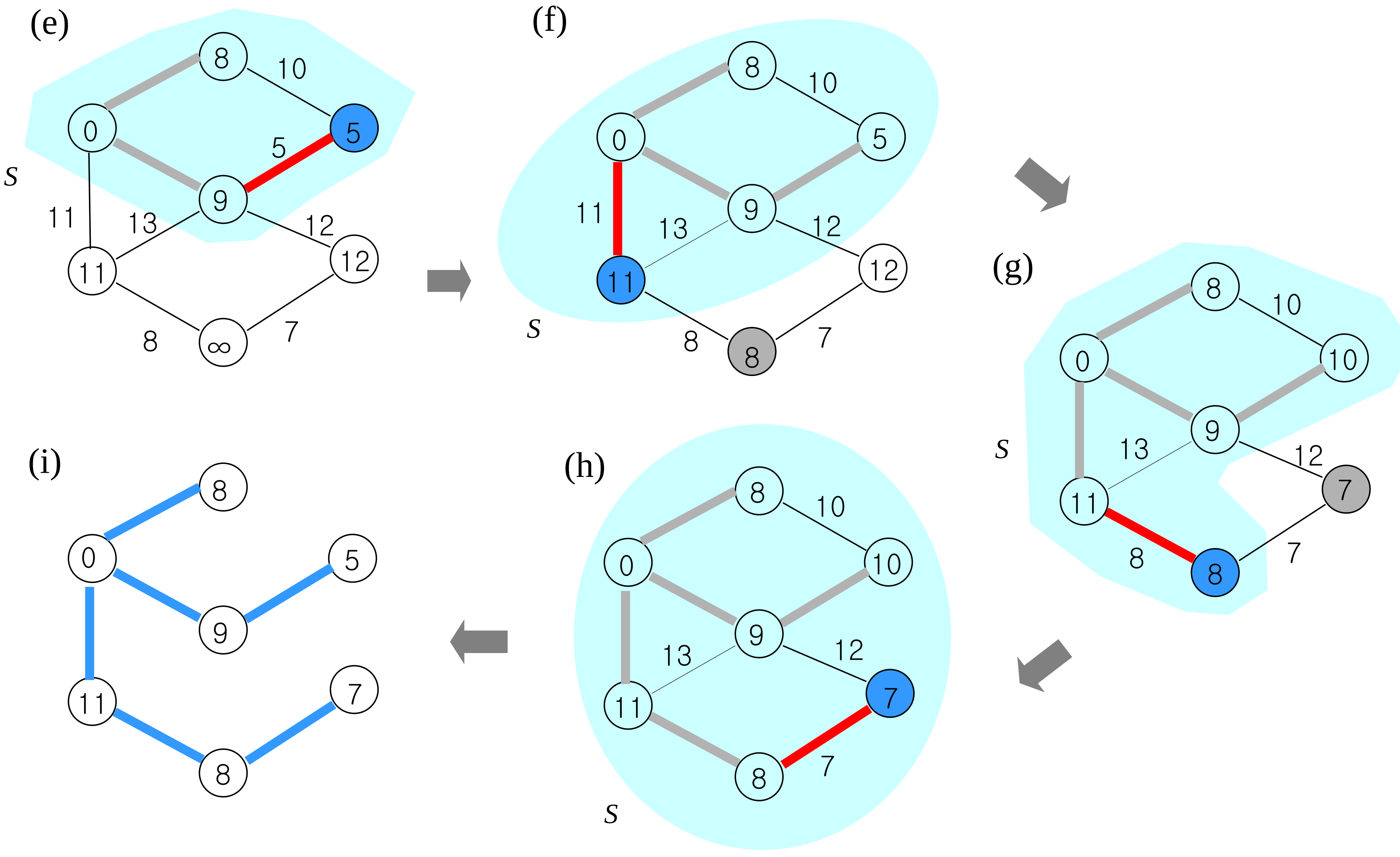
Heap 이용

Prim Algorithm의 작동 예



: 방금 S에 포함된 정점

: 방금 이완이 일어난 정점


(e)
S
8
10
0
5
5
9
11
13
12
11
12
8
∞
7
(f)
8
10
0
5
11
13
9
12
11
12
S
8
8
7
(g)
8
10
0
10
S
13
9
12
11
7
8
8
7
(h)
8
10
0
10
13
9
12
11
7
8
7
S
(i)
8
0
5
9
11
7
8

Prim Algorithm을 좀 더 구체적으로

Prim(G, r)
▷ G=(V, E): 주어진 그래프
▷ r: 시작으로 삼을 정점
{
    $S \leftarrow \Phi$;   ▷ S : 정점 집합
    for each $u \in V$
        $d_u \leftarrow \infty$;
    $d_r \leftarrow 0$;
    while ($S \neq V$){   ▷ n회 순환된다
        $x \leftarrow$ extractMin(V-S, d) ;
        $S \leftarrow S \cup \{x\}$;
        for each $v \in x$.adjList ▷ x.adjList : x로부터 연결된 정점 집합
            if ($v \in V$-S **and** $w_{x,v} < d_v$) **then** $d_v \leftarrow w_{x,v}$;
    }
}

이완(relaxation)

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

✓수행시간: $O(E\log V)$

힙 이용

Kruskal Algorithm

Kruskal (G, r)

{

1. $T \leftarrow \Phi$; ▷ T : 신장트리
2. 단 하나의 정점만으로 이루어진 n 개의 집합을 초기화한다;
3. 간선 집합 $Q(=E)$를 가중치가 작은 순으로 정렬한다;
4. **while** (T의 간선수 < n-1) {

    Q에서 최소비용 간선 (u, v)를 제거한다;

    if (정점 u와 정점 v가 서로 다른 집합에 속함) {

        두 집합을 하나로 합친다;

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

    }

}

}

✓수행시간: $O(E\log V)$

Kruskal Algorithm의 수행시간

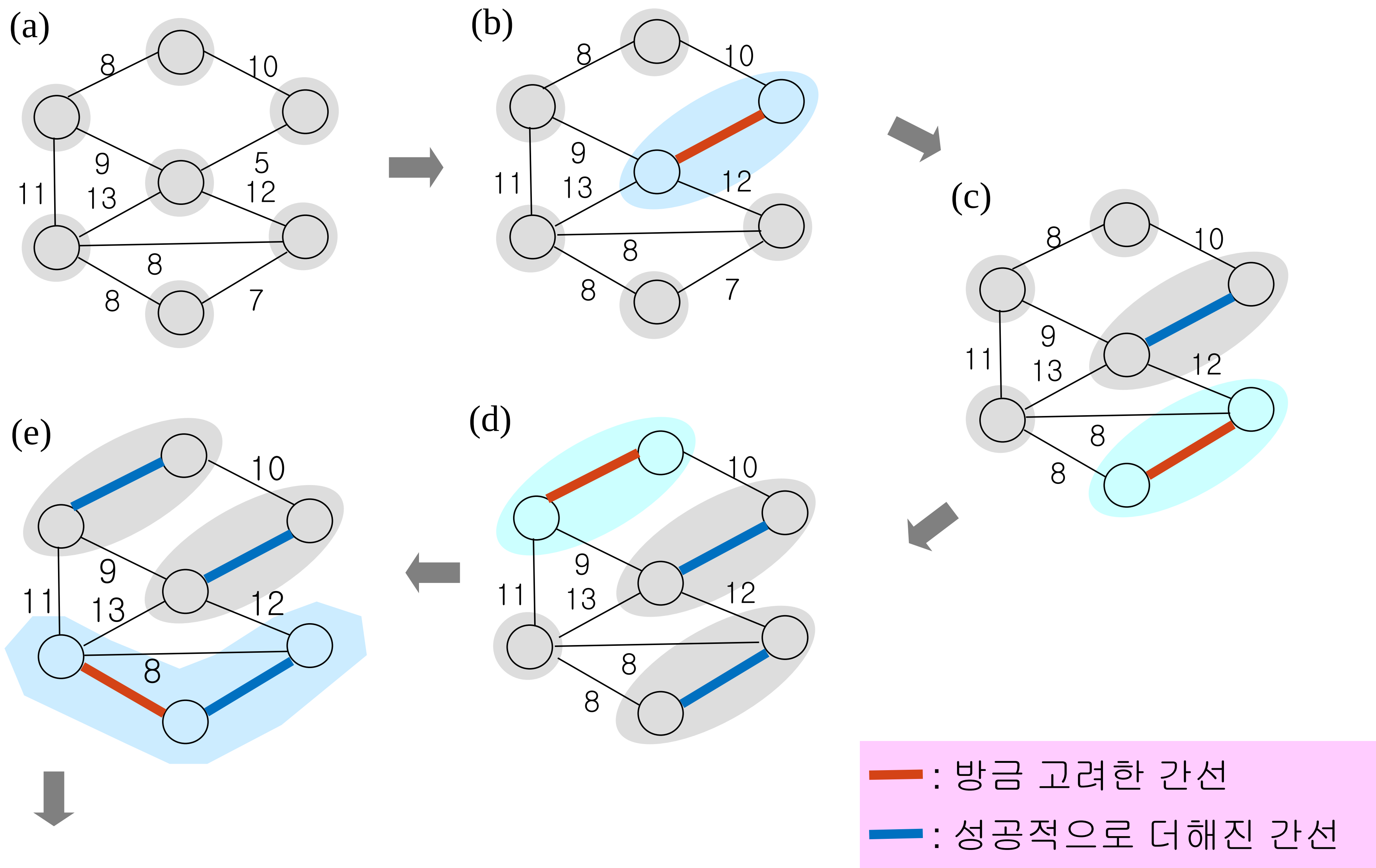
Step 2: $\Theta(V)$

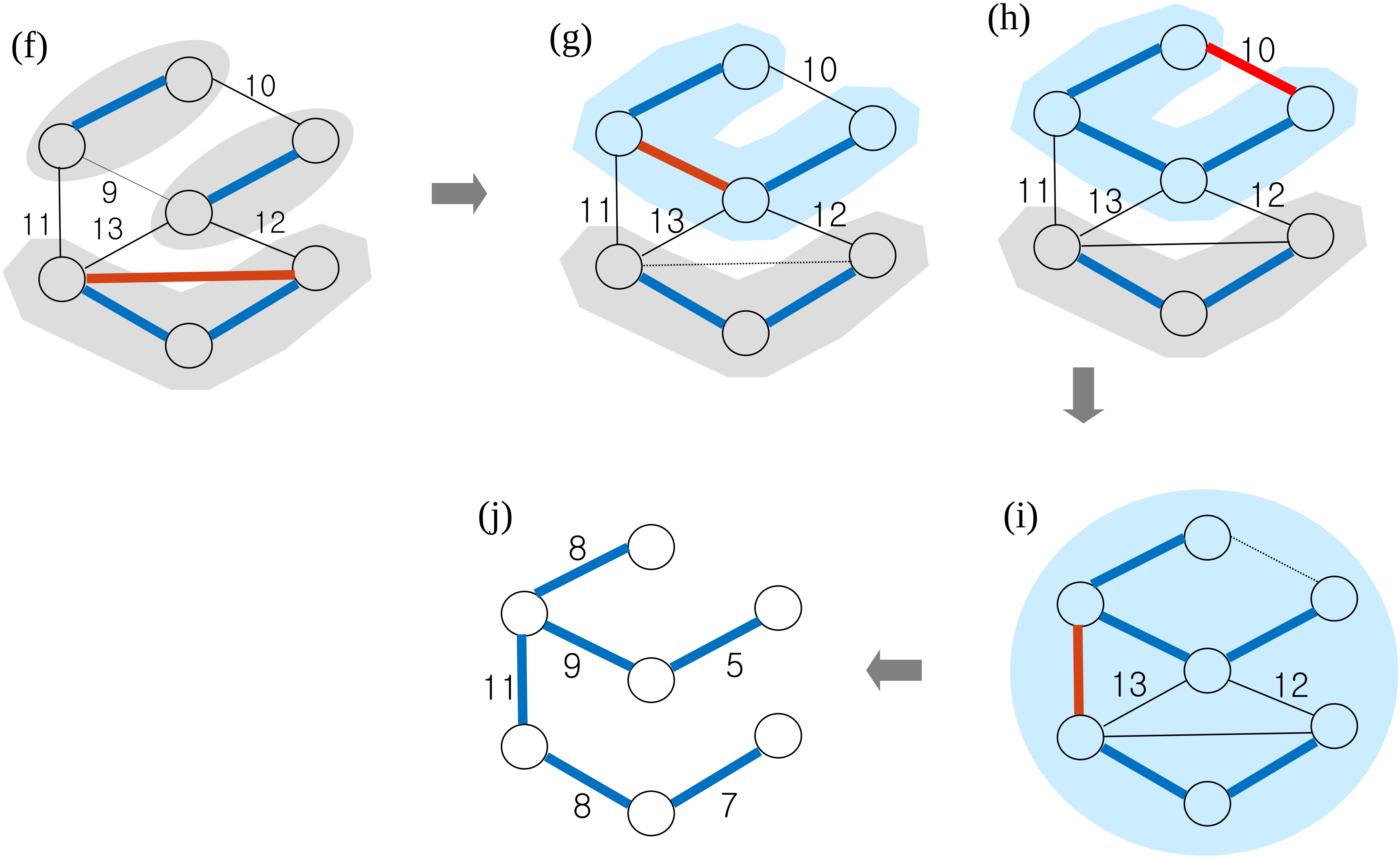
Step 3: $O(E\log E) = O(E\log V)$

Loop 4: $O(E\log^* V)$ by an efficient set handling

Totally $O(E\log V)$

Kruskal Algorithm의 작동 예




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

안정성 정리: Prim과 Kruskal Algorithm의 이론적 근거

[Theorem] 안정성 정리

Let (S,V-S) an arbitrary partition of vertices. Let {u,v} be the min-weight edge among those crossing S and V-S. Then there exists at least a min. spanning tree containing {u,v}.

<Proof> 뒷 페이지.

<Proof>

①

In any m.s.t. T not containing the edge $\{u, v\}$, there is only one path bet'n u & v. The path contains at least one edge crossing S & V-S. Take such a crossing edge $\{x, y\}$.

Note that $w_{uv} \le w_{xy}$. (given condition)

Case 1: $w_{uv} < w_{xy}$

$T \cup \{\{u, v\}\} - \{\{x, y\}\}$ is less weighted than T.

T is not a m.s.t. Contradiction to ①!

Case 2: $w_{uv} = w_{xy}$

$T \cup \{\{u, v\}\} - \{\{x, y\}\}$ has the same weight as T.

Hence $T \cup \{\{u, v\}\} - \{\{x, y\}\}$ is also a m.s.t.

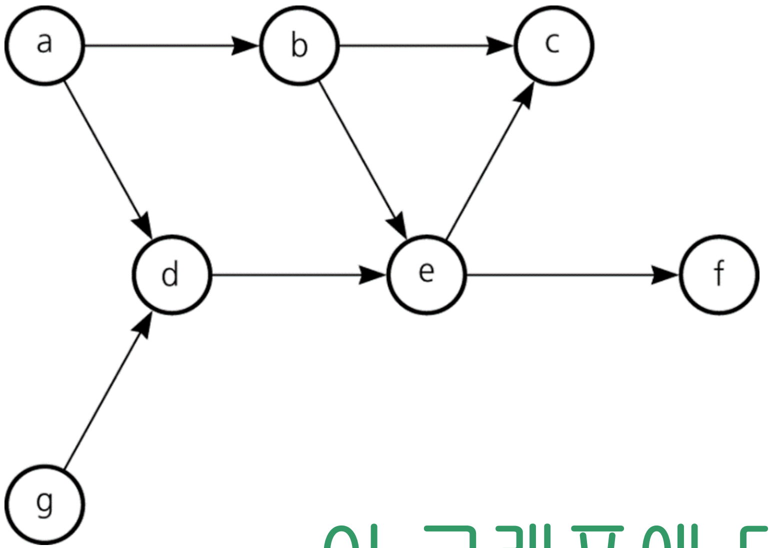
A Property

If edge weights are all distinct, then there is only one m.s.t.

If not, there can be more than one m.s.t.

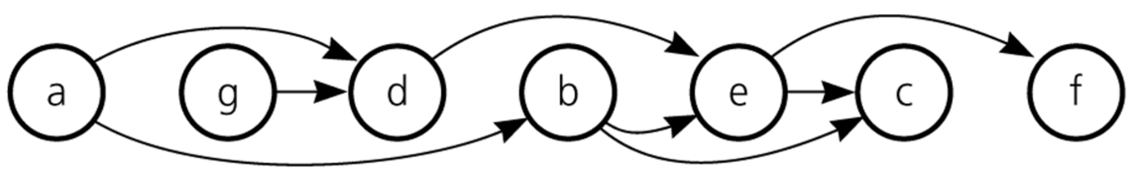
Topological sorting

- 조건
  - 싸이클이 없는 유향 그래프 (DAG, Directed Acyclic Graph)
- 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 순서topological order가 존재한다

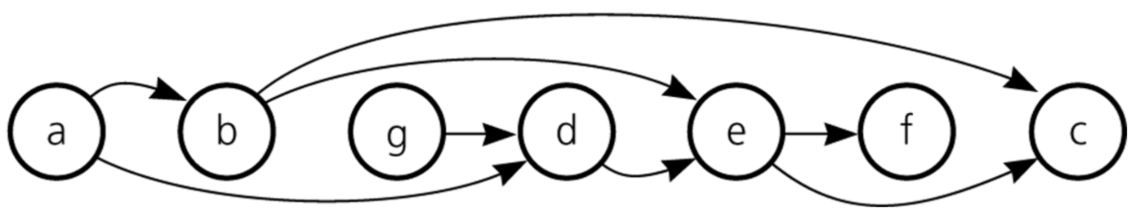


이 그래프에 대한 위상정렬의 예 2개

(a)



(b)

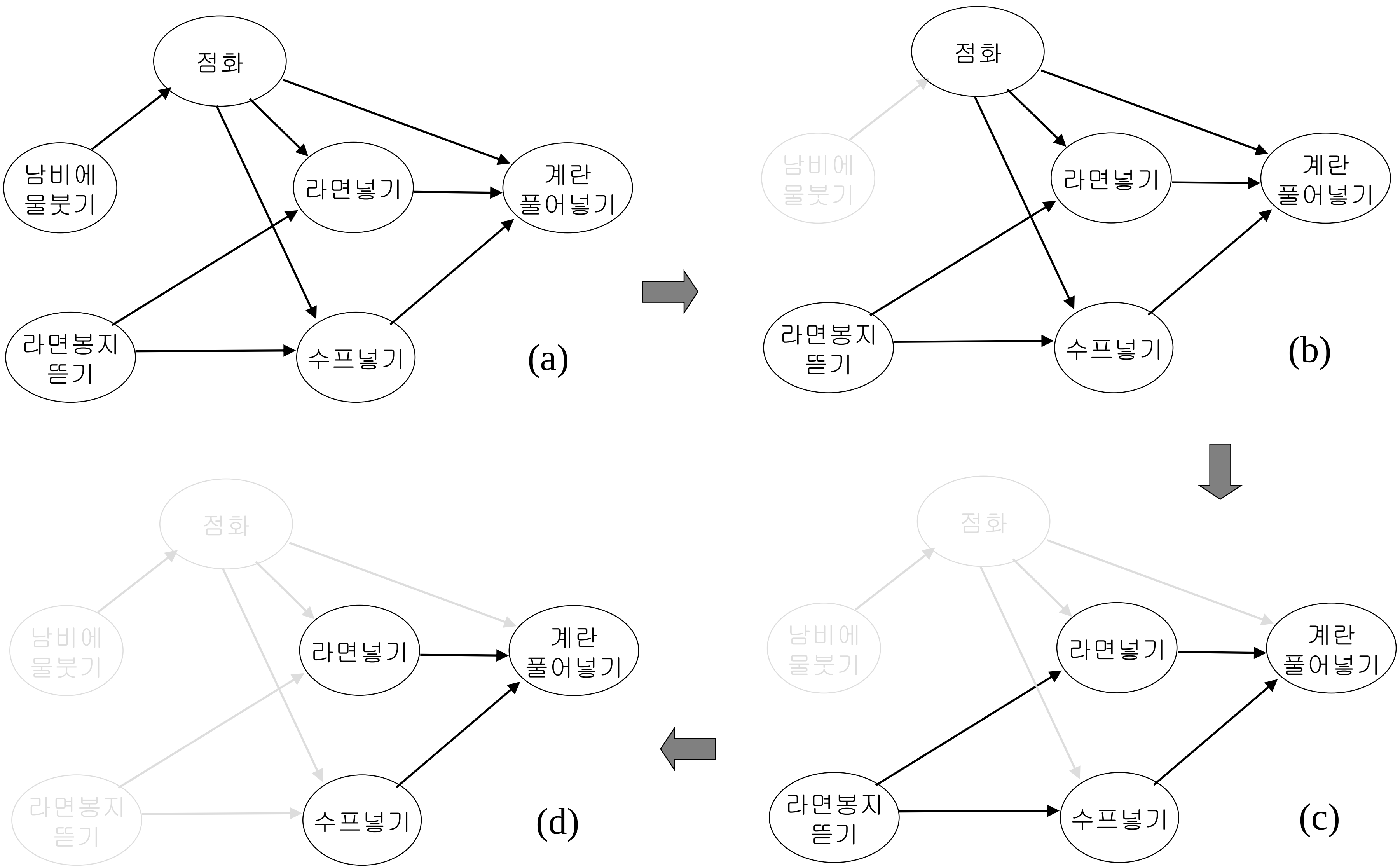


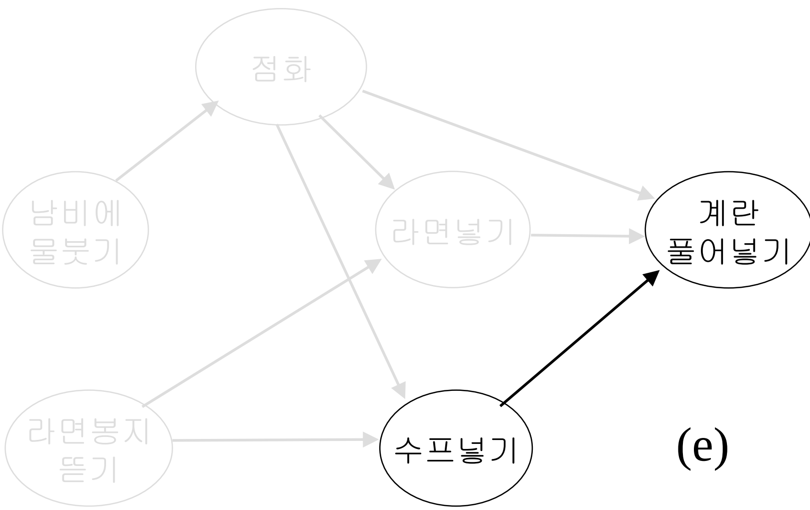
위상정렬 알고리즘 1

topologicalSort1(G)
{
    for $i \leftarrow 1$ **to** n {
        진입간선incoming edge이 없는 정점 u를 선택한다;
        $A[i] \leftarrow u$;
        정점 u와, u의 진출간선outgoing edge을 모두 제거한다;
    }
    ▷ 이 시점에 배열 $A[1 \ldots n]$에는 정점들을 위상정렬 되어 있다
}

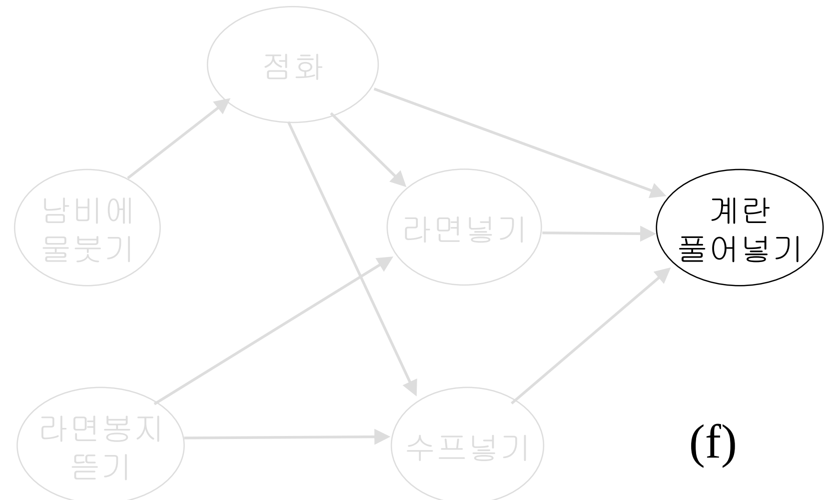
✓수행시간: $\Theta(V+E)$

위상정렬 알고리즘 1의 작동 예

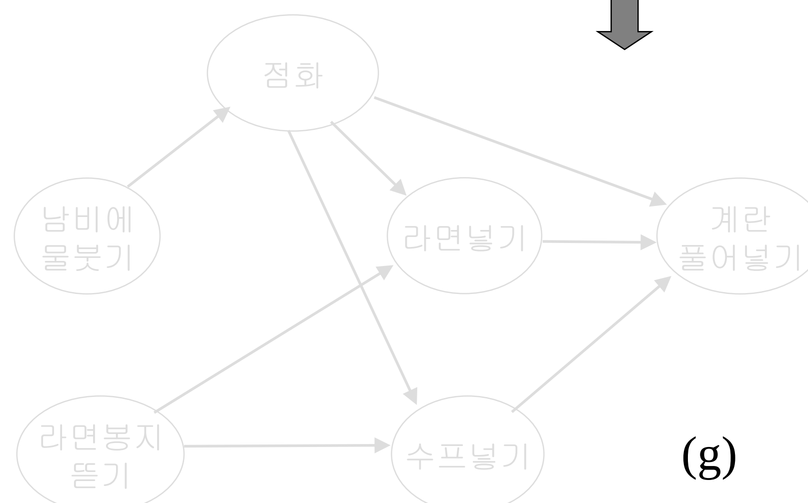



점화
남비에
물붓기
라면넣기
계란
풀어넣기
라면봉지
뜯기
수프넣기

(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 $i \leftarrow n$ **downto** 1 {
    진출간선outgoing edge이 없는 정점 u를 선택한다;
    $A[i] \leftarrow u$;
    정점 u와, u의 진입간선incoming edge을 모두 제거한다;
  }
  ▷ 이 시점에 배열 $A[1\ldots n]$에는 정점들을 위상정렬 되어 있다
}

✓수행시간: $\Theta(V+E)$

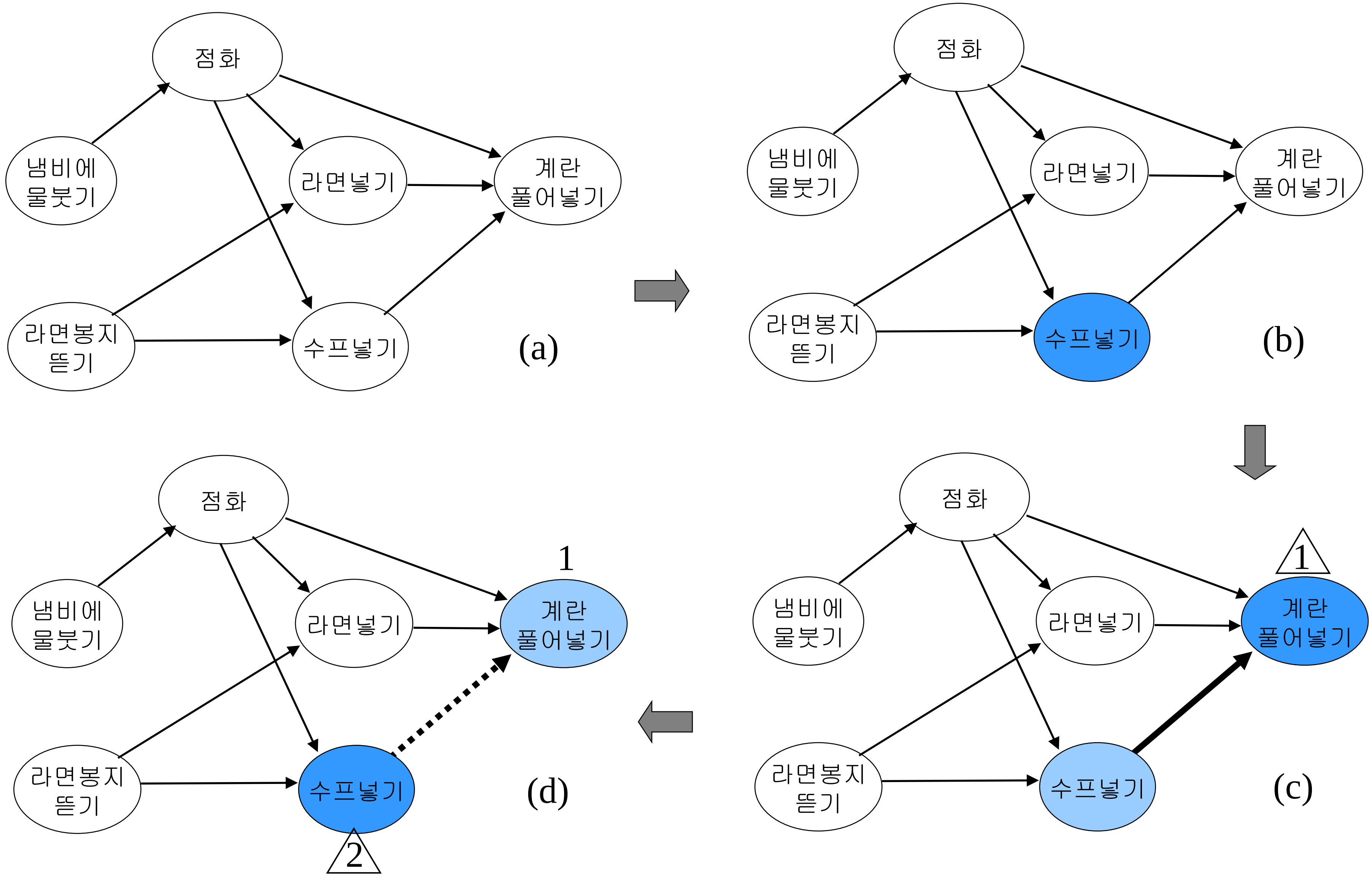
위상정렬 알고리즘 3

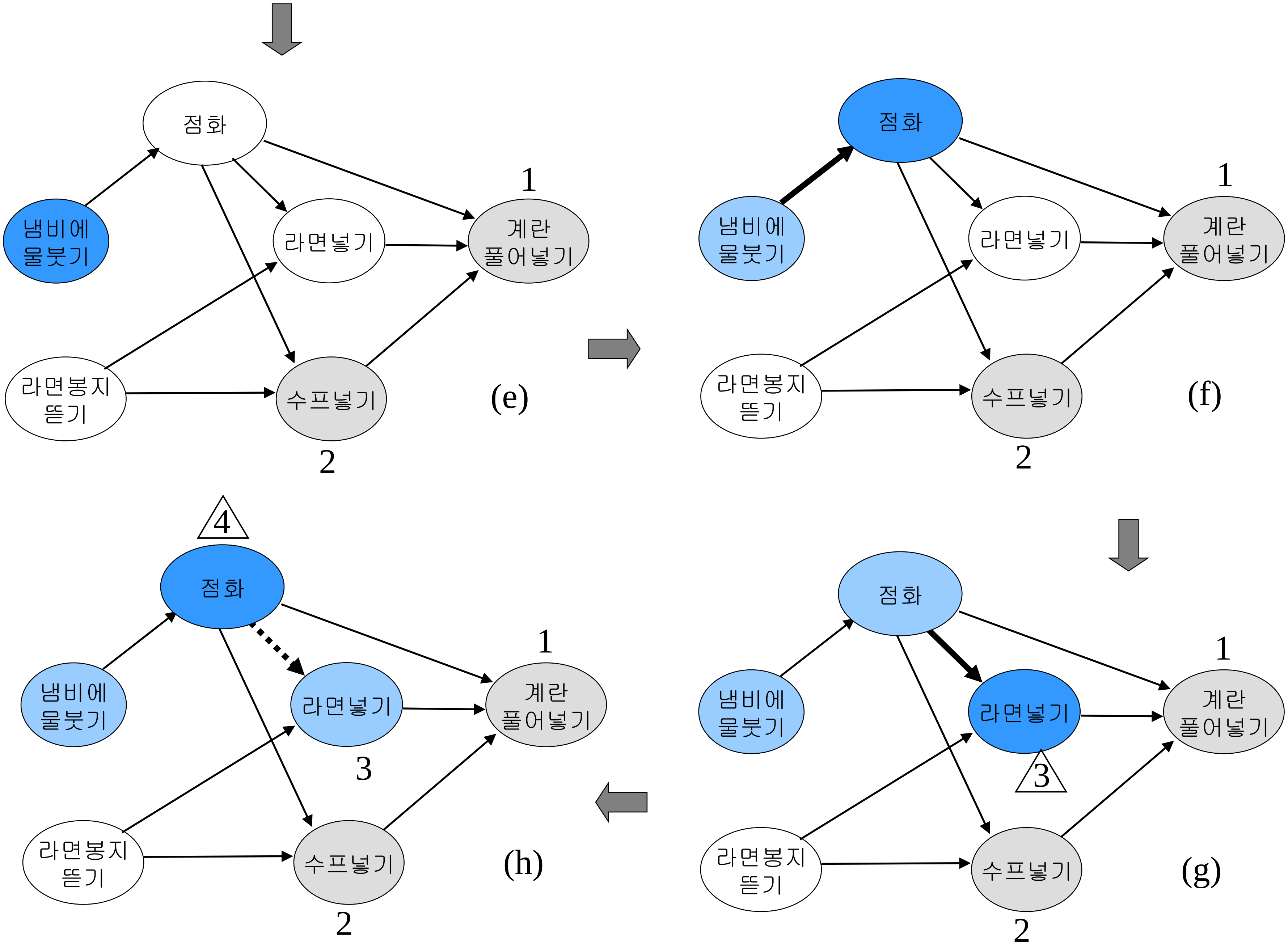
topologicalSort3(G)
// DFS-TS(v)가 방문하지 못한 vertex가 있으면 이들 중 아무거나로부터
시작하여 또 DFS-TS를 수행한다. 이를 끝날 때까지 반복.
{
for each $v \in V$
v.visited ← NO;
for each $v \in V$ ▷ 정점의 순서는 아무 순서나 무관
if (v.visited = NO) **then** DFS-TS(v) ;
}

✓수행시간: $\Theta(V+E)$

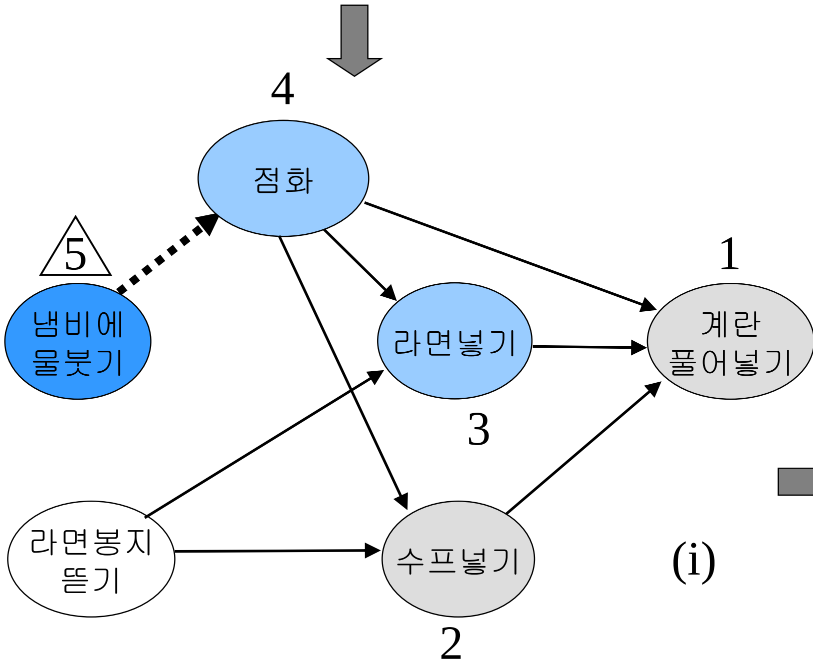
DFS-TS(v)
// 알고리즘이 끝나고 나면 연결 리스트 R에는 정점들이 위상정렬된 순서로 매달린다.
{
v.visited ← YES;
for each $x \in v$.AdjList ▷ v.AdjList: v 로부터 연결된 정점 집합
if (x.visited = NO) **then** DFS-TS(x) ;
연결 리스트 R의 맨 앞에 정점 v를 삽입한다;
}

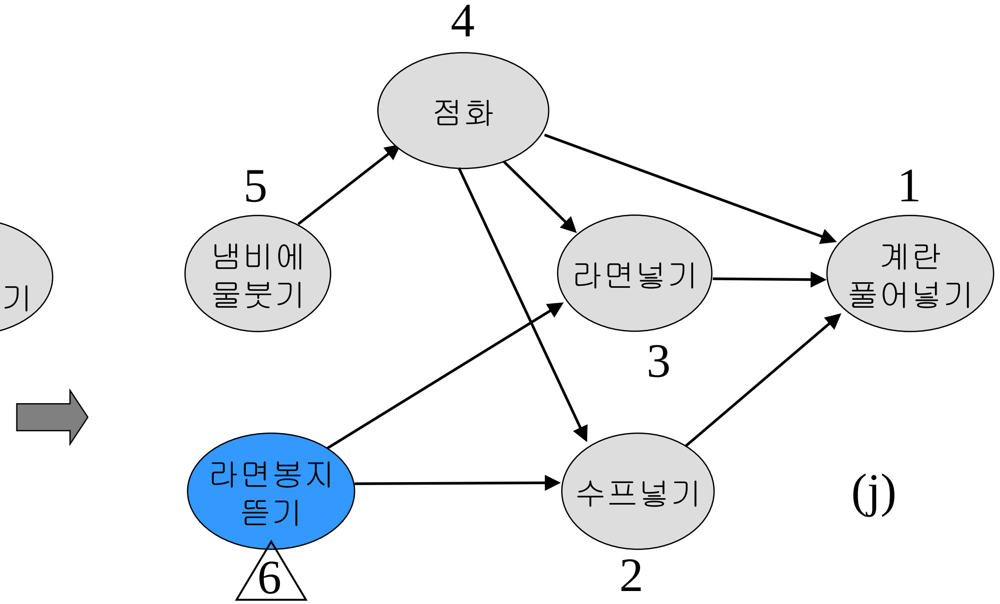
위상정렬 알고리즘 3의 작동 예





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





1 … 6 : finishing time

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 (single-source shortest path)
  - 단일 시작점으로부터 각 정점에 이르는 최단경로를 구한다
  - Dijkstra 알고리즘
    - 음의 가중치를 허용하지 않는 최단경로
  - Bellman-Ford 알고리즘
    - 음의 가중치를 허용하는 최단경로
  - 싸이클이 없는 그래프의 최단경로
- 모든 쌍 최단경로 (all-pairs shortest path)
  - 모든 정점 쌍 사이의 최단경로를 모두 구한다
  - Floyd-Warshall 알고리즘

Dijkstra Algorithm

모든 간선의 가중치는 음이 아니어야 함

Dijkstra(G, r)
▷ G=(V, E): 주어진 그래프
▷ r: 시작으로 삼을 정점
{
  $S \leftarrow \Phi$;   ▷ S : 정점 집합
  for each $u \in V$
    $d_u \leftarrow \infty$;
  $d_r \leftarrow 0$;
  while ($S \neq V$){   ▷ n회 순환된다
    $x \leftarrow$ extractMin(V-S, d) ;
    $S \leftarrow S \cup \{x\}$;
    for each $v \in x$.adjList ▷ x.adjList : x로부터 연결된 정점 집합
      if ($v \in V$-S **and** $d_x + w_{x,v} < d_v$) **then** $d_v \leftarrow d_x + w_{x,v}$;
  }
}

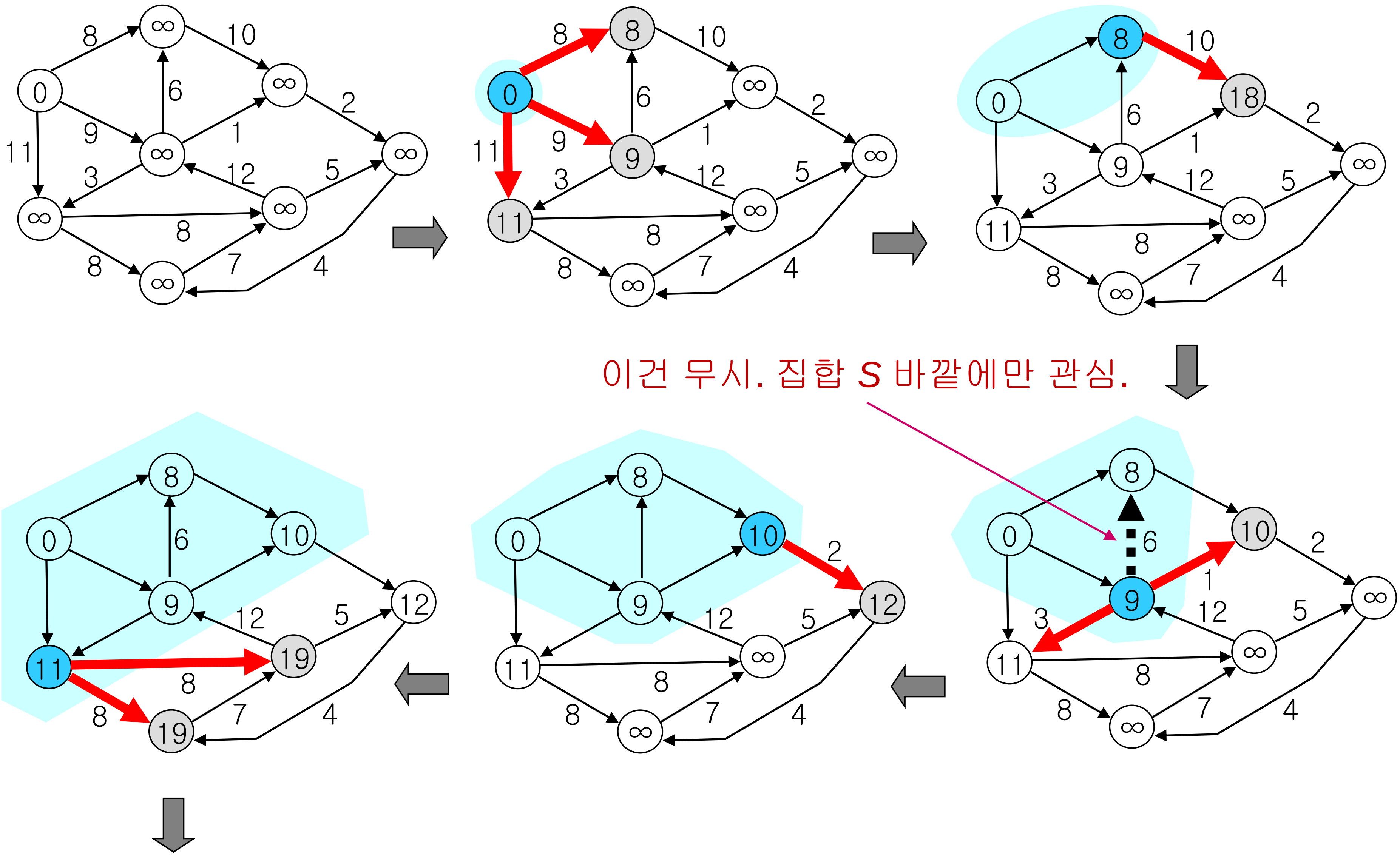
이완(relaxation)

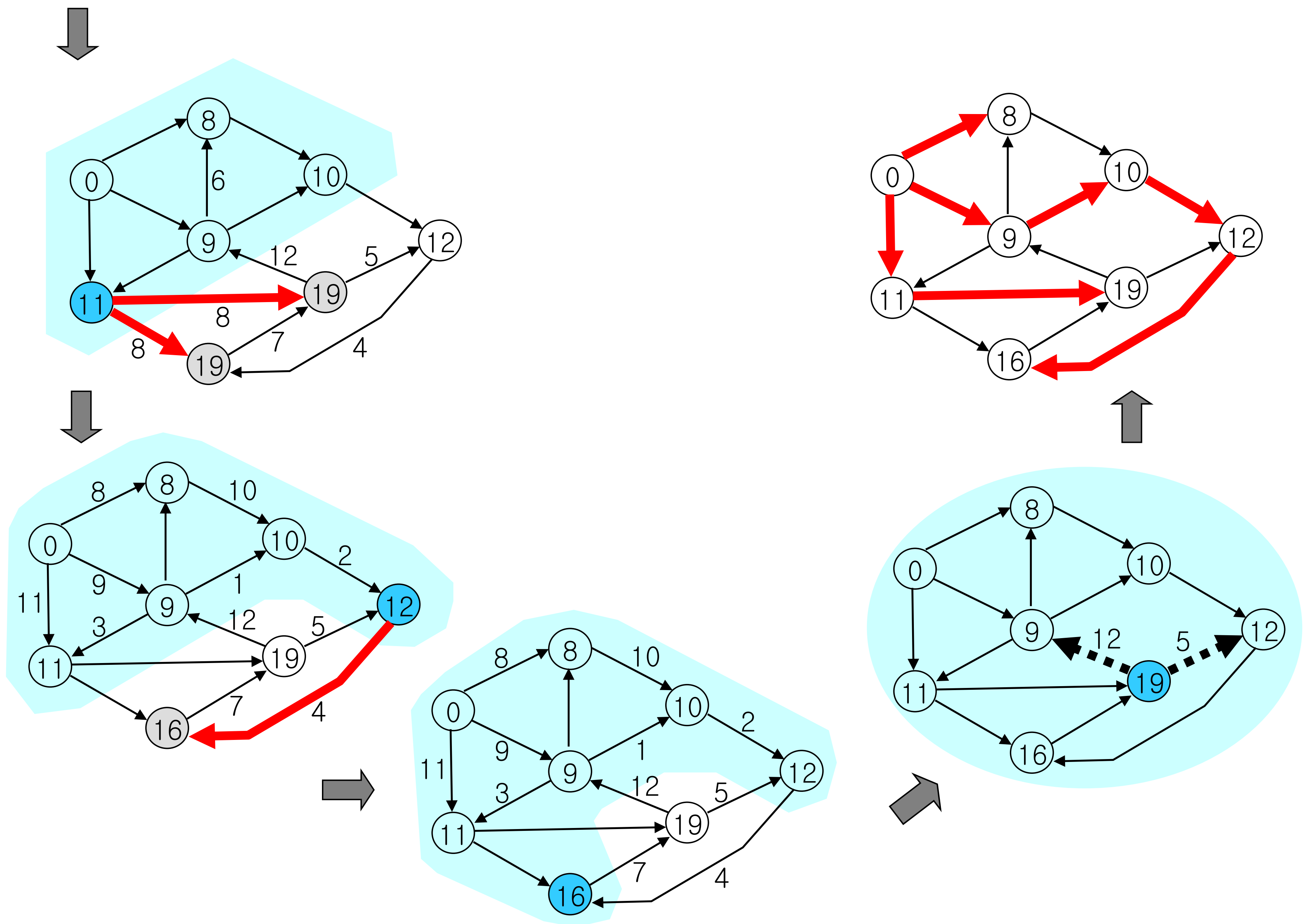
extractMin(Q, d)
{
  집합 Q에서 d값이 가장 작은 정점 x를 리턴한다 ;
}

✓수행시간: $O(E\log V)$

힙 이용

Dijkstra Algorithm의 작동 예





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

Bellman-Ford Algorithm

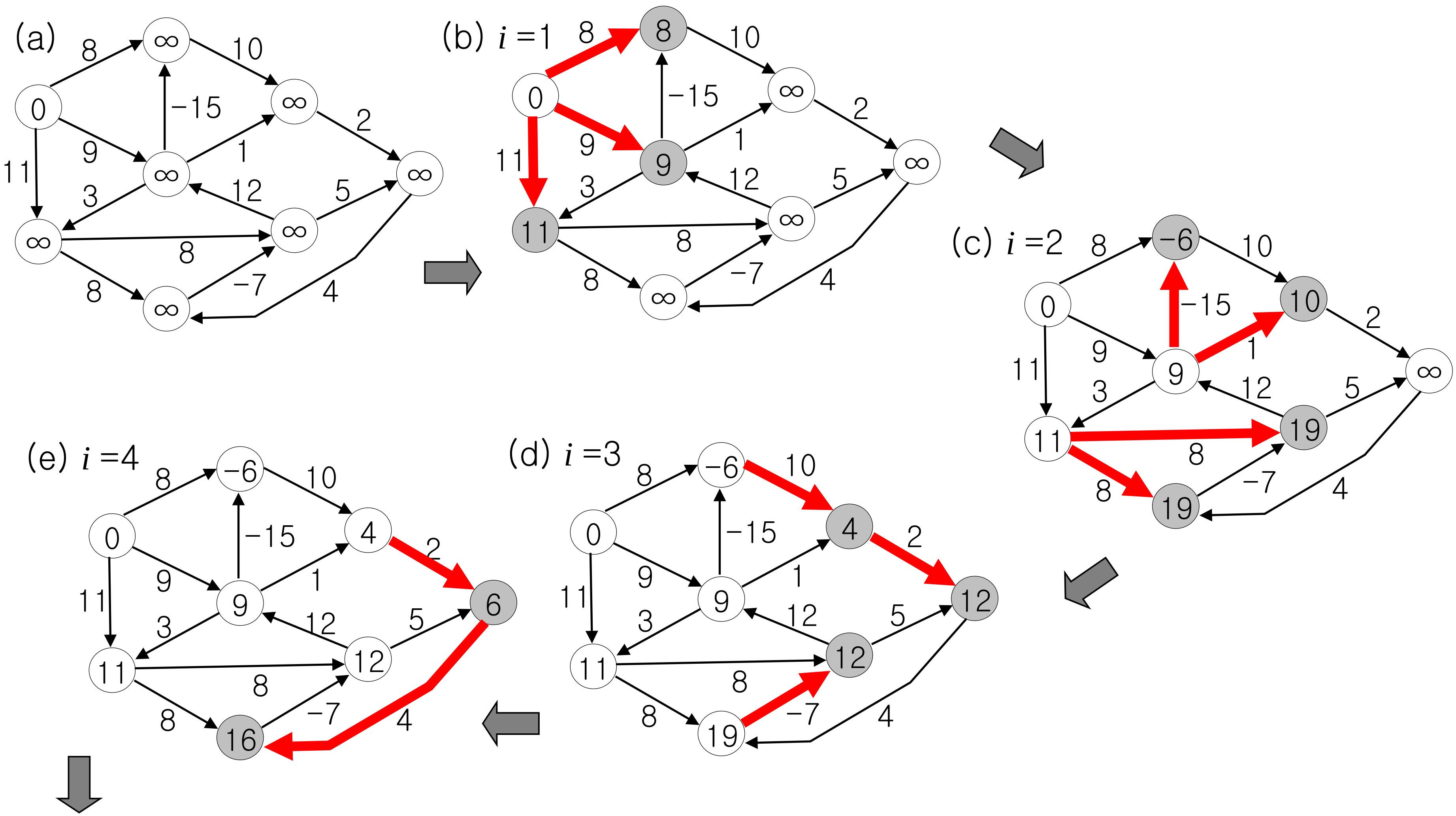
음의 가중치를 허용한다

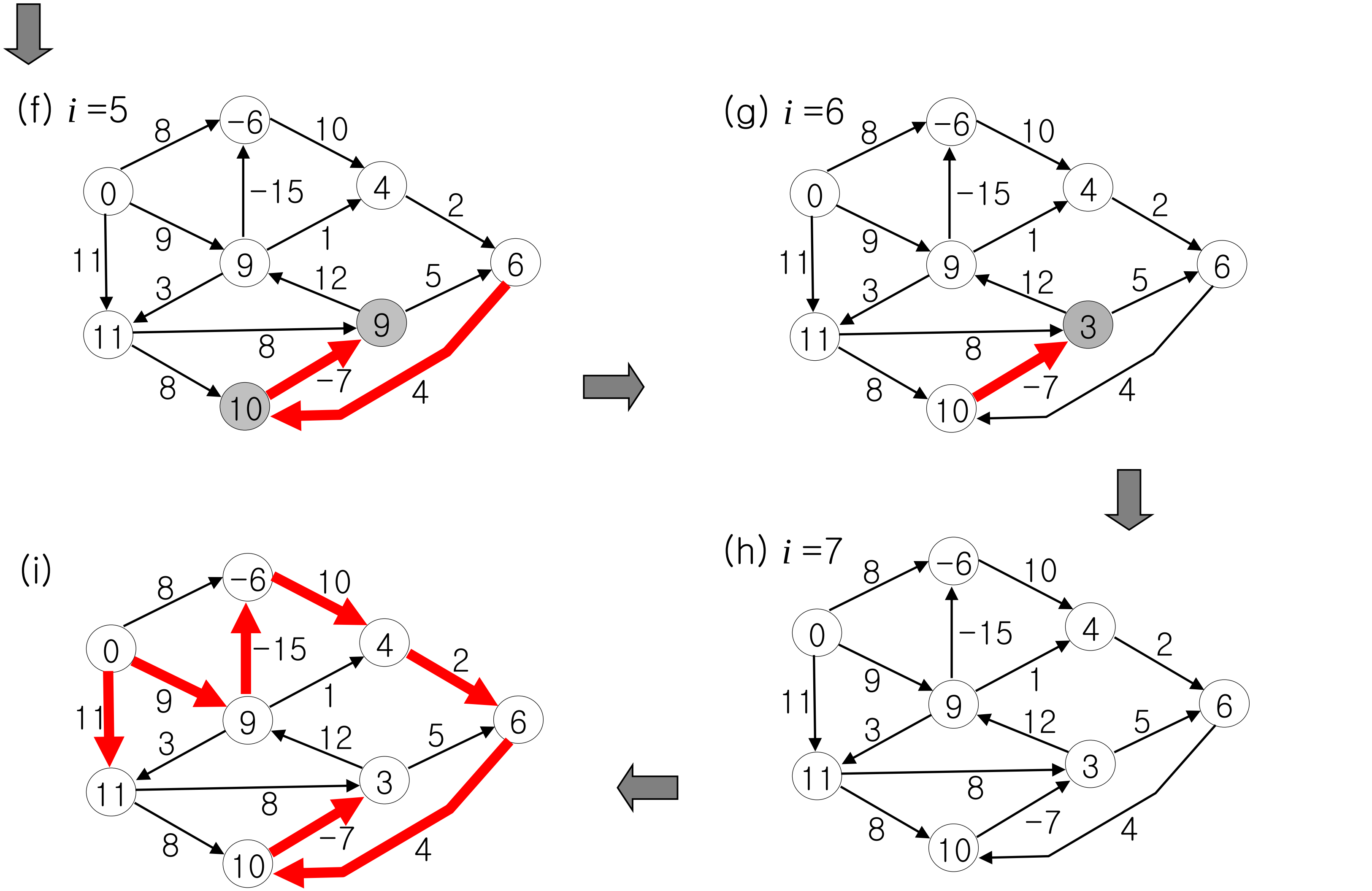
BellmanFord(G, r)
{
  for each $u \in V$
    $d_u \leftarrow \infty$;
  $d_r \leftarrow 0$;
  for $i \leftarrow 1$ **to** $|V|$-1
    for each $(u, v) \in E$
      if $(d_u + w_{u,v} < d_v)$ **then** $d_v \leftarrow d_u + w_{u,v}$;
}

이완(relaxation)

✓수행시간: $\Theta(EV)$
✓$O(EV)$가 되도록 할 수 있다

Bellman-Ford Algorithm의 작동 예




(f) i =5
(g) i =6
(h) i =7
(i)

DP로 본 Bellman-Ford 알고리즘

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

✓ 재귀적 관계

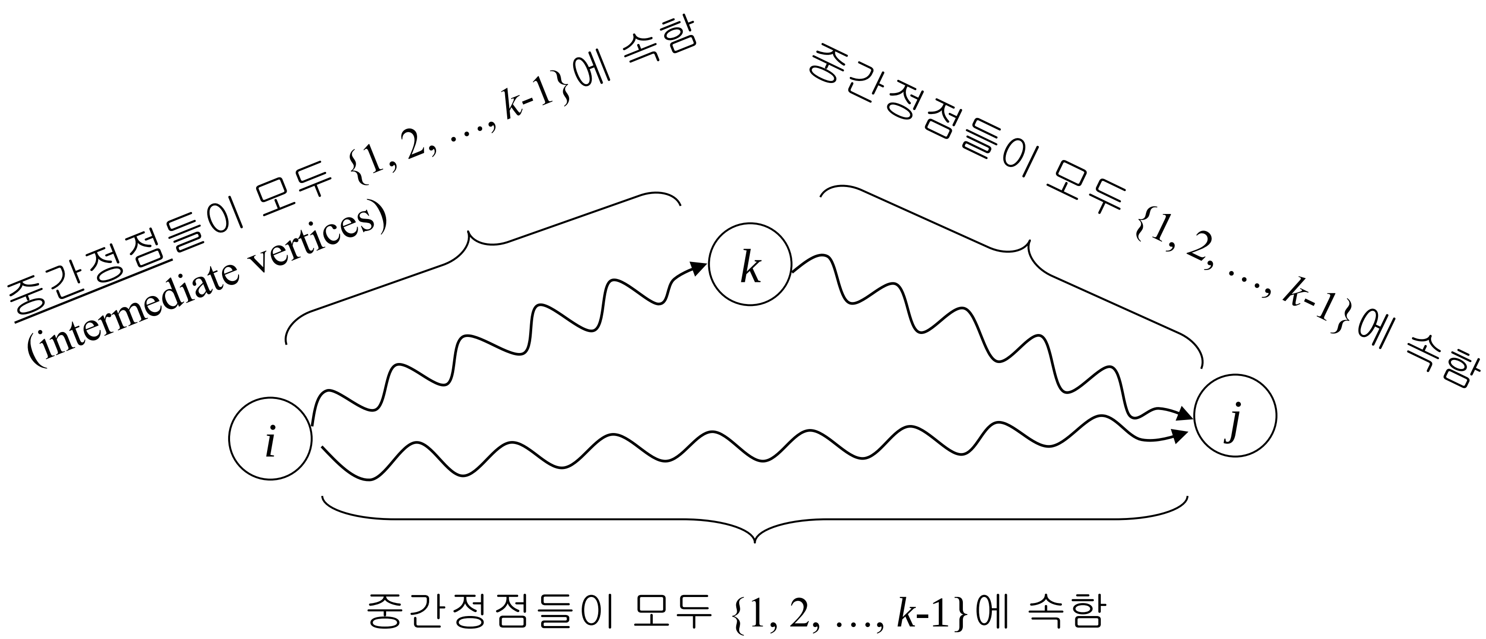
$$
\begin{cases}
d_v^k = \min\limits_{(u,v) \in E} \{d_u^{k-1} + w_{u,v}\}, \quad k > 0 \\
d_r^0 = 0 \\
d_t^0 = \infty, \quad t \neq r
\end{cases}
$$

Floyd-Warshall Algorithm

- 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

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



Floyd-Warshall Algorithm

FloydWarshall(G)
{
  for $i \leftarrow 1$ **to** n
    for $j \leftarrow 1$ **to** n
      $d_{ij}{}^{0} \leftarrow w_{ij}$;
  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
        $d_{ij}{}^{k} \leftarrow \min \{d_{ij}{}^{k\text{-}1} , d_{ik}{}^{k\text{-}1} + d_{kj}{}^{k\text{-}1}\}$;
}

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

DAG의 Shortest Path

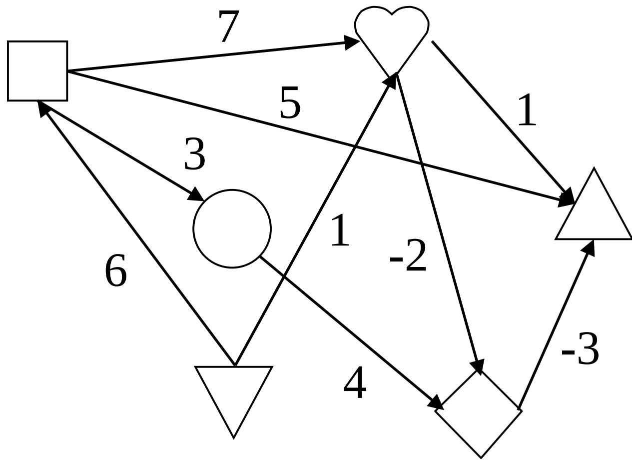
- 사이클이 없는 유향 그래프
  - DAG: Directed Acyclic Graph
- DAG에서의 최단경로는 선형시간에 간단히 구할 수 있다

DAG-ShortestPath(G, s)
{
for each $u \in V$
$d_u \leftarrow \infty$;
$d_s \leftarrow 0$;
G의 정점들을 위상정렬한다;
for each $x \in V$ in topological order starting at s
for each $v \in x$.adjList ▷ x.adjList : x로부터 연결된 정점 집합
if $(d_x + w_{x,v} < d_v)$ **then** $d_v \leftarrow d_x + w_{x,v}$;
}

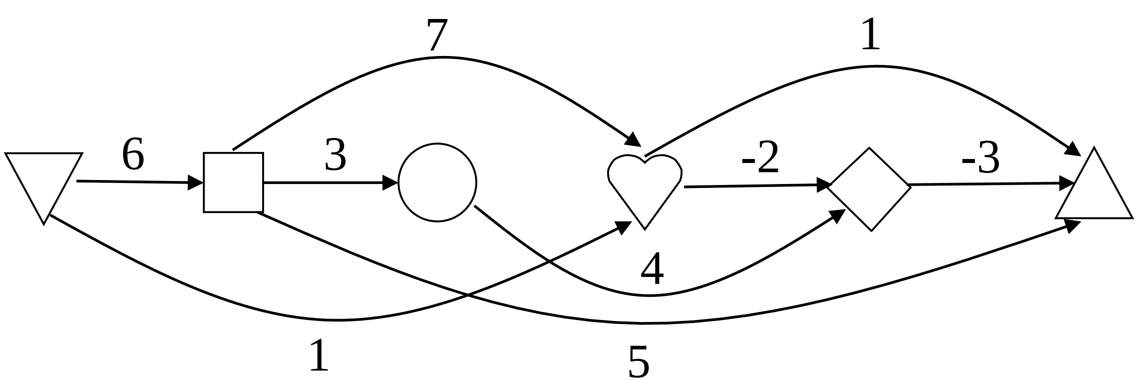
✓수행시간: $\Theta(V+E)$

DAG-ShortestPath의 작동 예

(a)

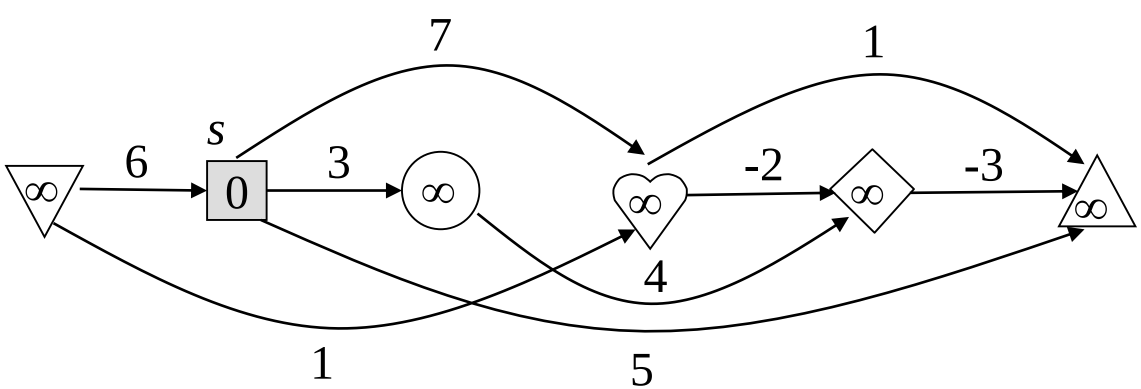


(b)



Topological sorting

(c)

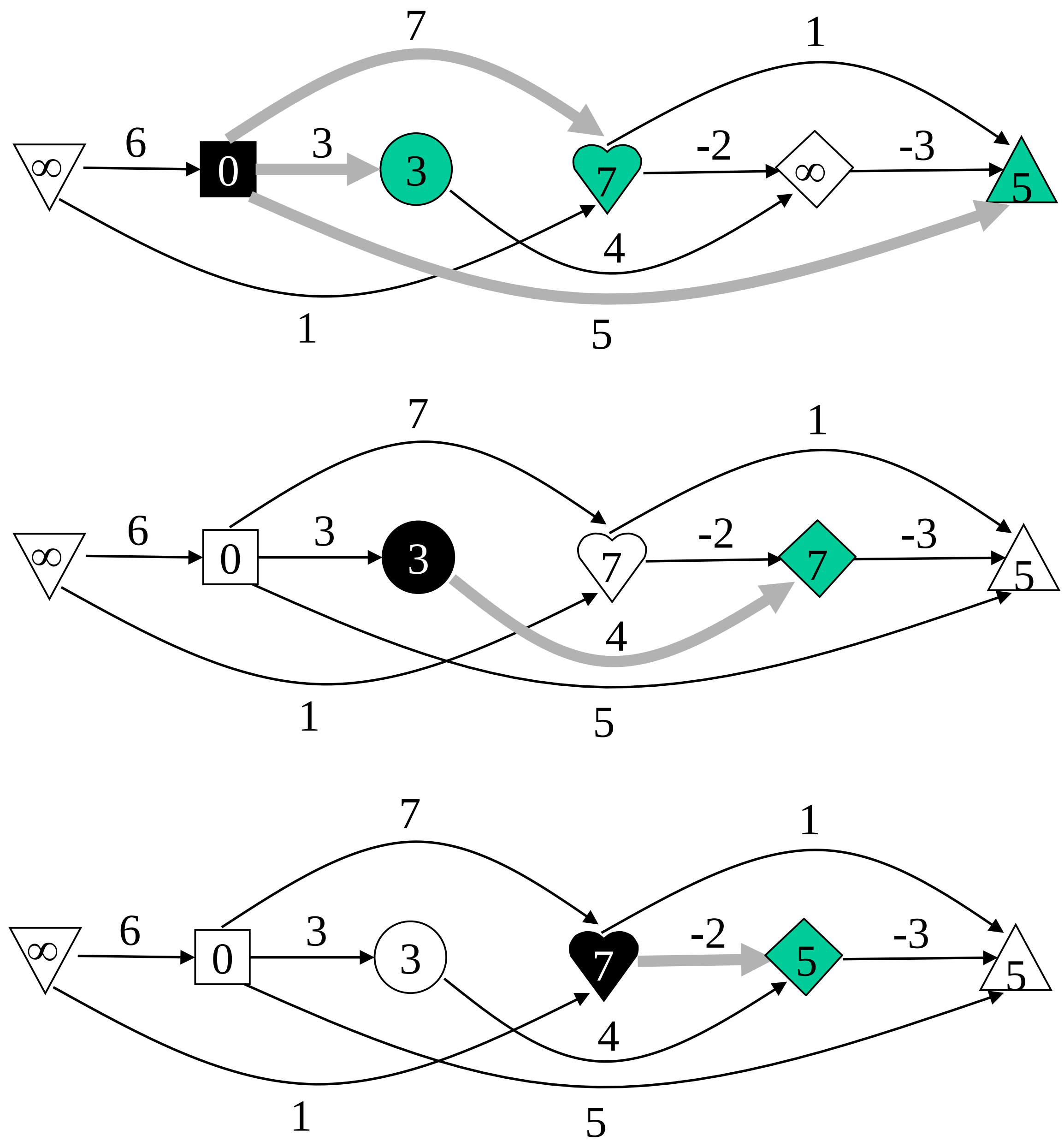


Initialization

(d)

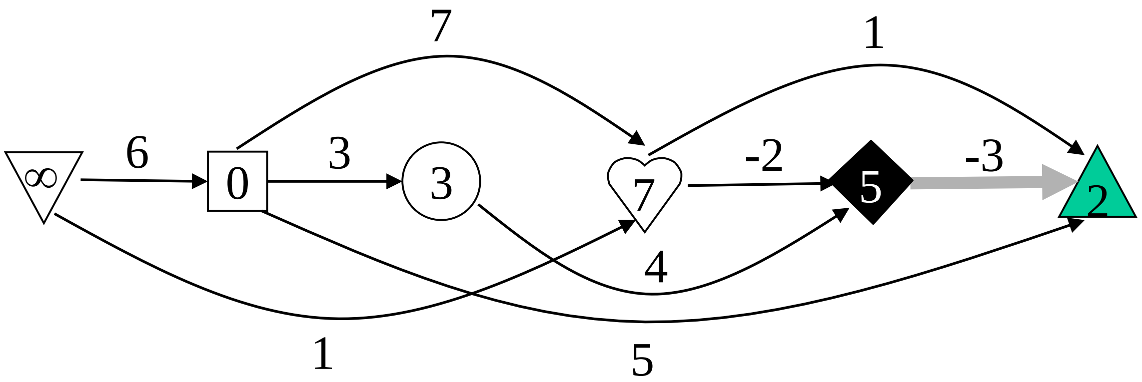
(e)

(f)

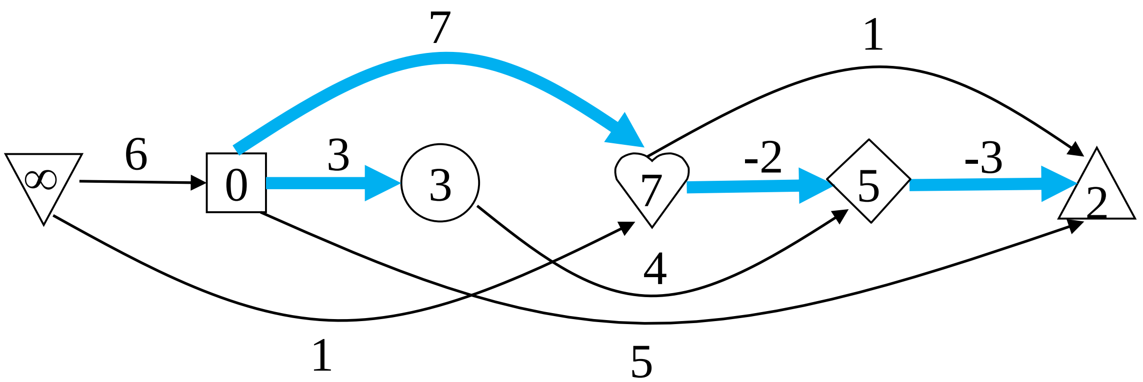


Start relaxation

(g)



(h)



Strongly Connected Components

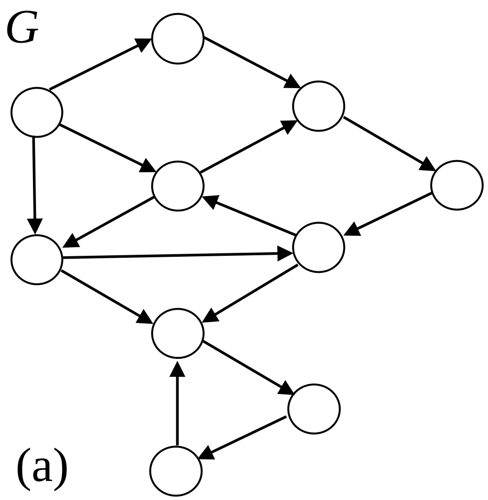
- 강하게 연결됨
  - Digraph의 모든 정점쌍에 대해서 양방향으로 경로가 존재하면 강하게 연결되었다고 한다
  - 강하게 연결된 부분 그래프를 강연결요소Strongly connected component라 한다
- 임의의 그래프에서 강연결요소들을 찾는다

SCCs 구하기 알고리즘

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

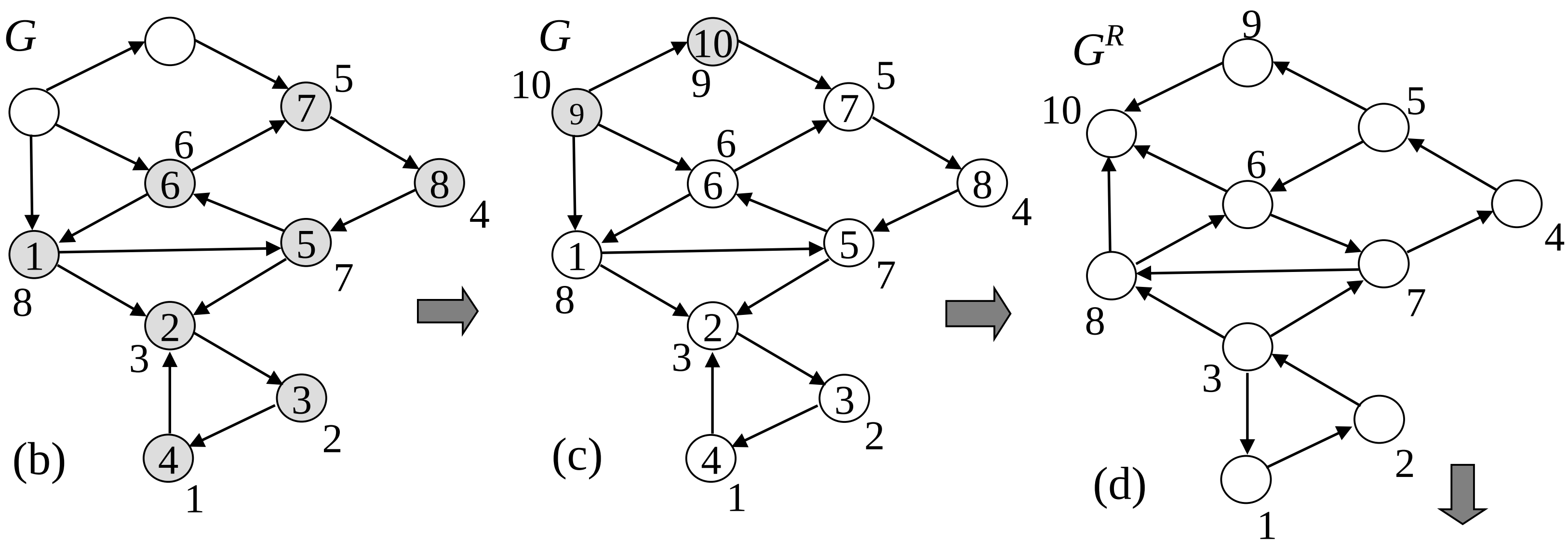
✓수행시간: $\Theta(V+E)$

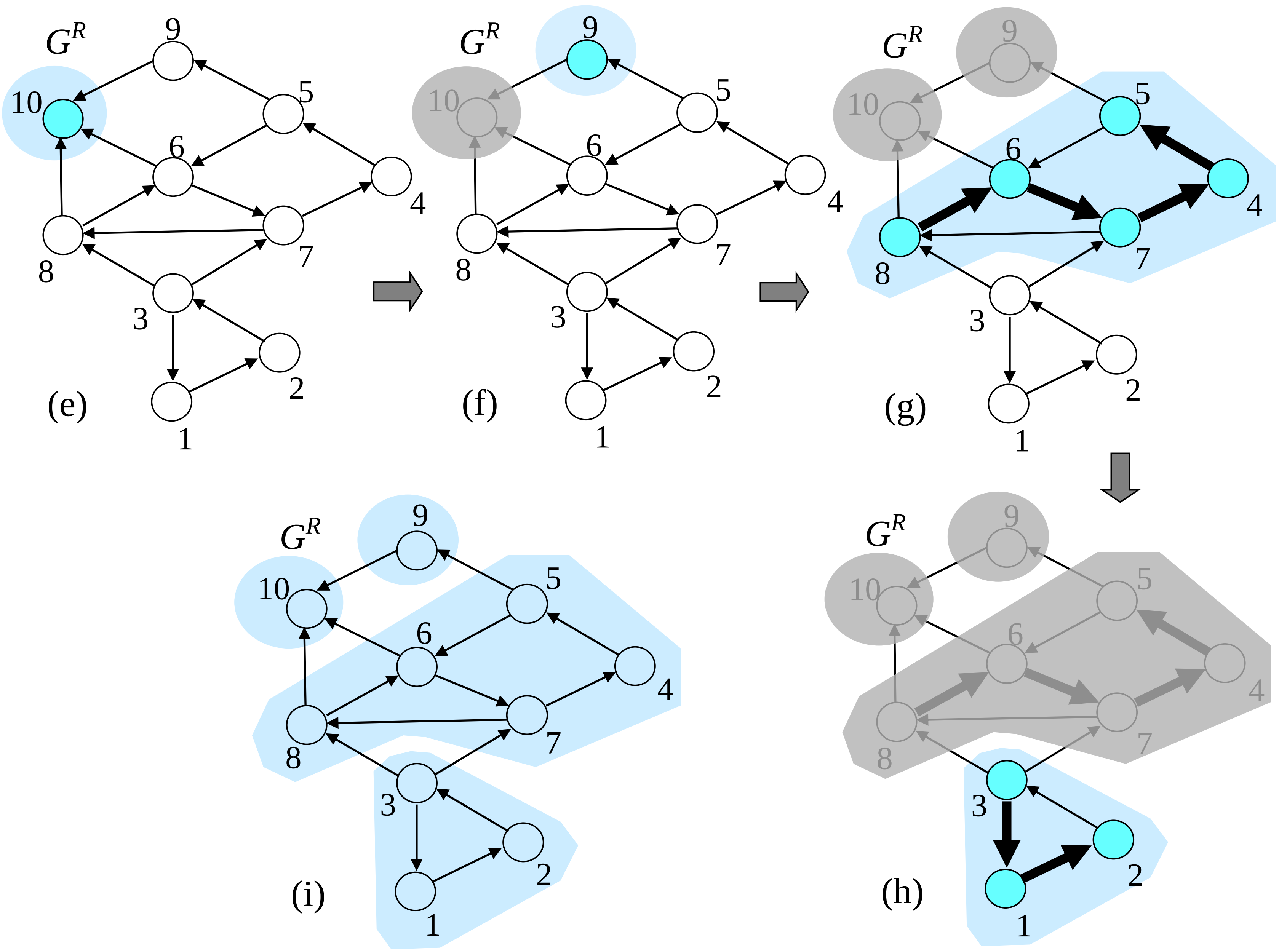
stronglyConnectedComponent의 작동 예



7 : 방문순서

5 : 해당 노드의 finishing time




G^R
(e)
(f)
(g)
(h)
(i)
1
2
3
4
5
6
7
8
9
10

Proof

[Claim]

v & w are in the same strongly connected component

iff

v & w are in the same tree in the DFS forest of step 3

<Proof>

$\Rightarrow$ easy (trivial)

Assume v & w are in the same strongly connected component.

Then $\exists(v \rightarrow^{+} w)$ and $\exists(w \rightarrow^{+} v)$ in G. Hence $\exists(w \rightarrow^{+} v)$ and $\exists(v \rightarrow^{+} w)$ in G.

Suppose DFS in G^R starts at some vertex x and reaches v (or w),

then it also reaches w(or v); i.e., they are in the same tree in the forest.

$\Leftarrow$ Assume v & w are in the same tree in the forest of G^R

Then $\exists(x \rightarrow^{+} v)$ in G^R $\Rightarrow$ $\exists(v \rightarrow^{+} x)$ in G.

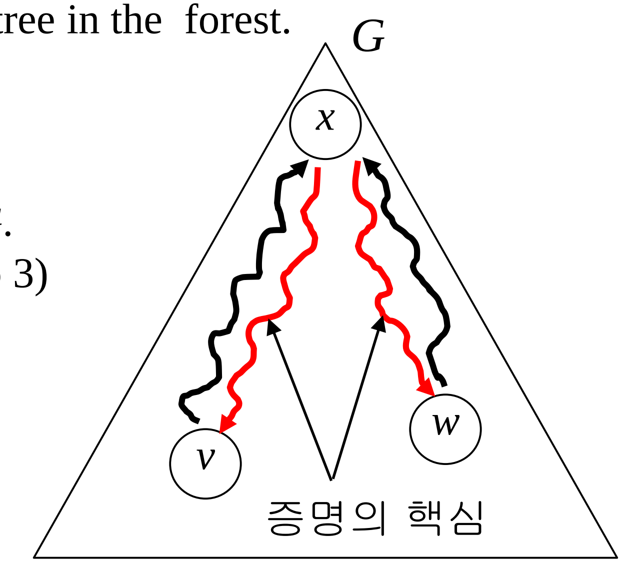
Suppose, for the contradiction, that $\exists$no path $x \rightarrow^{+} v$ in G.

Then, $f[x] < f[v]$, contradiction! (Since $f[x] > f[v]$ by step 3)

Similarly, we can show that $\exists(x \rightarrow^{+} w)$ in G.

Therefore $\exists(v \rightarrow^{+} w)$ and $\exists(w \rightarrow^{+} v)$ in G.

v & w are in the same strongly connected component.



∎