알고리즘

10장 그래프(graph)

학습내용

1. 그래프
2. 그래프의 표현
3. 너비 우선 탐색(BFS)과 깊이 우선 탐색(DFS)
4. 최소 신장 트리
5. 위상 정렬
6. 최단 경로
7. 강연결 요소

학습목표

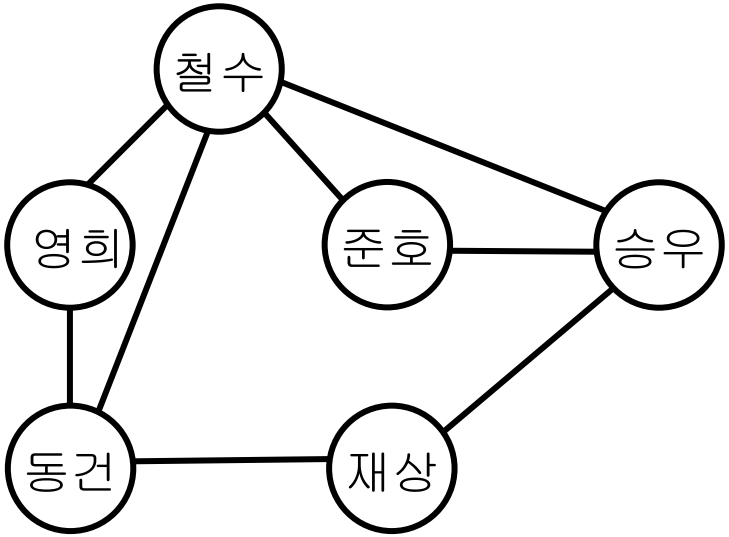
- 그래프의 표현법을 익힌다.
- 너비우선탐색과 깊이우선탐색의 원리를 충분히 이해한다.
- 신장트리의 의미와 최소신장트리를 구하는 두 가지 알고리즘을 이해한다.
- 그래프의 특성에 따라 가장 적합한 최단경로 알고리즘을 선택할 수 있도록 한다.
- 위상정렬을 이해하고 DAG의 경우에 위상정렬을 이용해 최단경로를 구하는 방법을 이해한다.
- 강연결 요소를 구하는 알고리즘을 이해한다.
- 각 알고리즘의 수행시간을 분석할 수 있도록 한다.

학습내용

1. **그래프**
2. 그래프의 표현
3. 너비 우선 탐색(BFS)과 깊이 우선 탐색(DFS)
4. 최소 신장 트리
5. 위상 정렬
6. 최단 경로
7. 강연결 요소

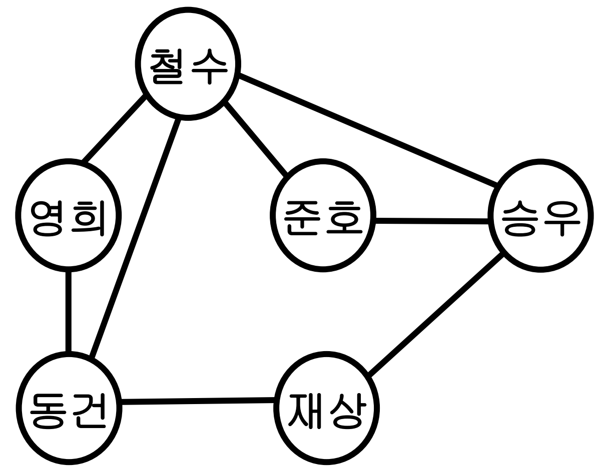
그래프(graph)

- 그래프란?
  - 현상이나 사물을 **정점(vertex)**과 **간선(edge)**으로 표현한 것
  - 정점은 대상을 나타내고, 간선은 대상들 간의 관계를 나타냄
- 예) 사람들 간의 친분관계를 나타낸 그래프



그래프의 정의

- 그래프 *G* = (*V*, *E*)
  - *V*: 정점 집합
  - *E*: 간선 집합



- 앞의 그래프 예를 다시 살펴보자.

G = (V, E)

V = {철수, 영희, 동건, 준호, 승우, 재상}

E = {(철수, 영희), (철수, 동건), (철수, 준호), (철수, 승우),
(영희, 동건), (동건, 재상), (준호, 승우), (재상, 승우)}

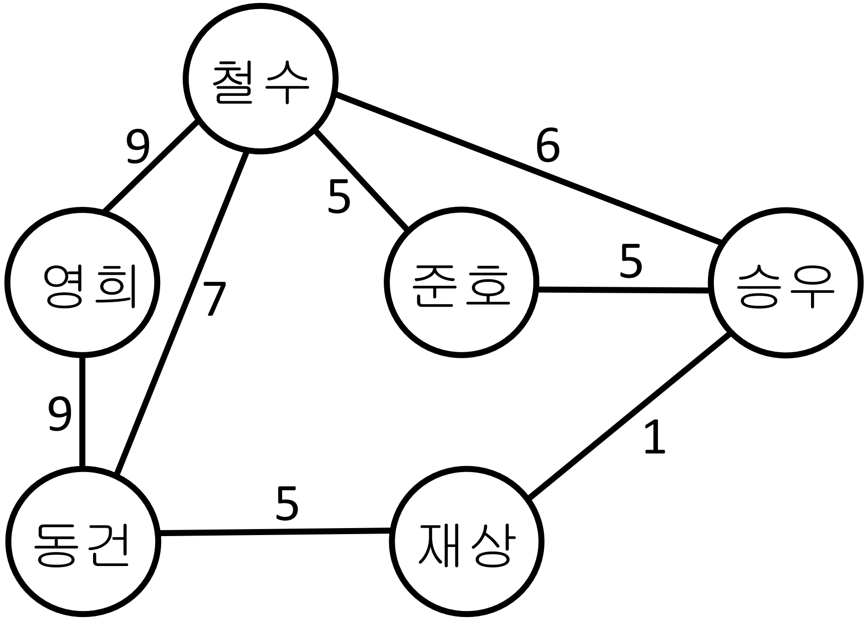
|V| = 6 // 집합 V의 원소 수, 즉 정점 수

|E| = 8 // 집합 E의 원소 수, 즉 간선 수

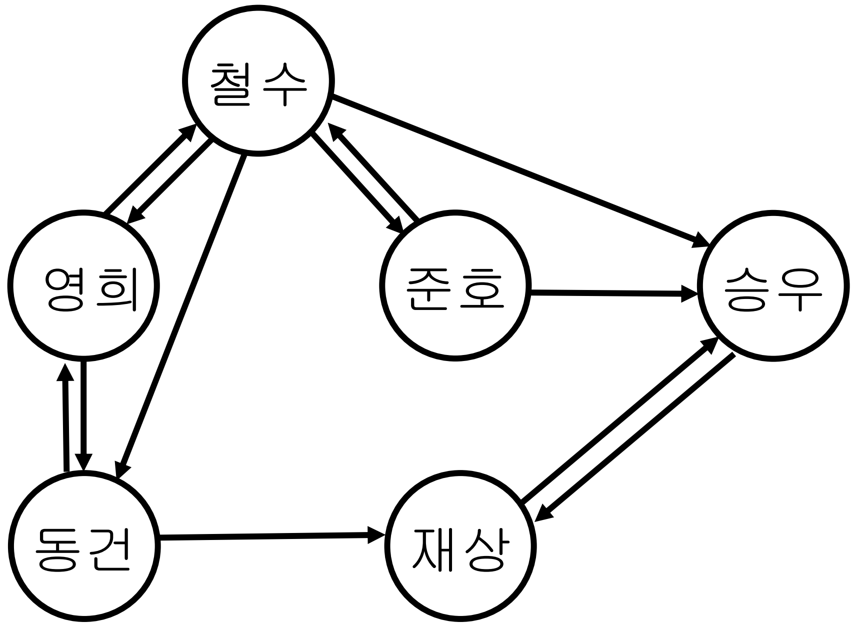
V(G) : 그래프 G의 vertex 집합

E(G) : 그래프 G의 edge 집합

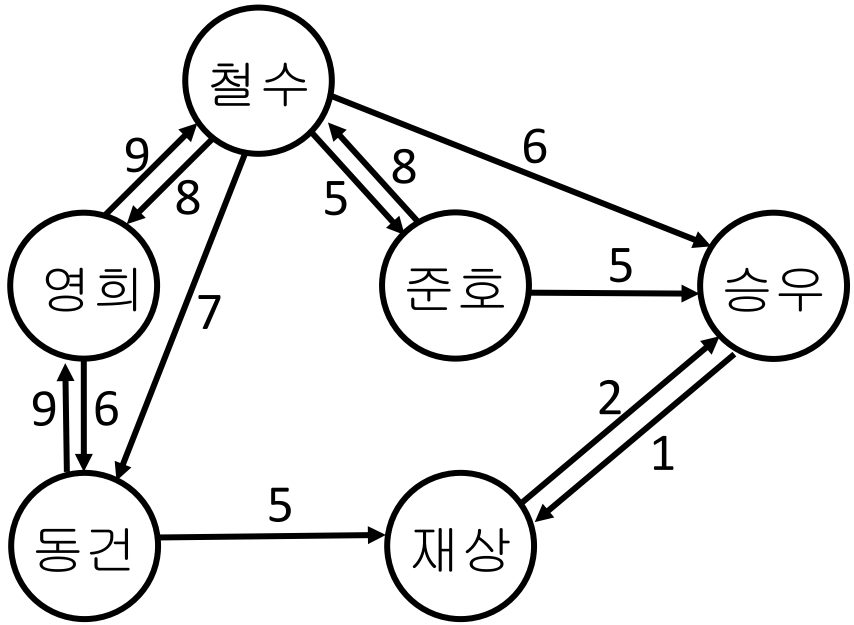
친분관계 그래프 – 친밀도를 가중치(weight)로 나타낸 경우



친분관계 그래프 – 방향을 고려한 경우

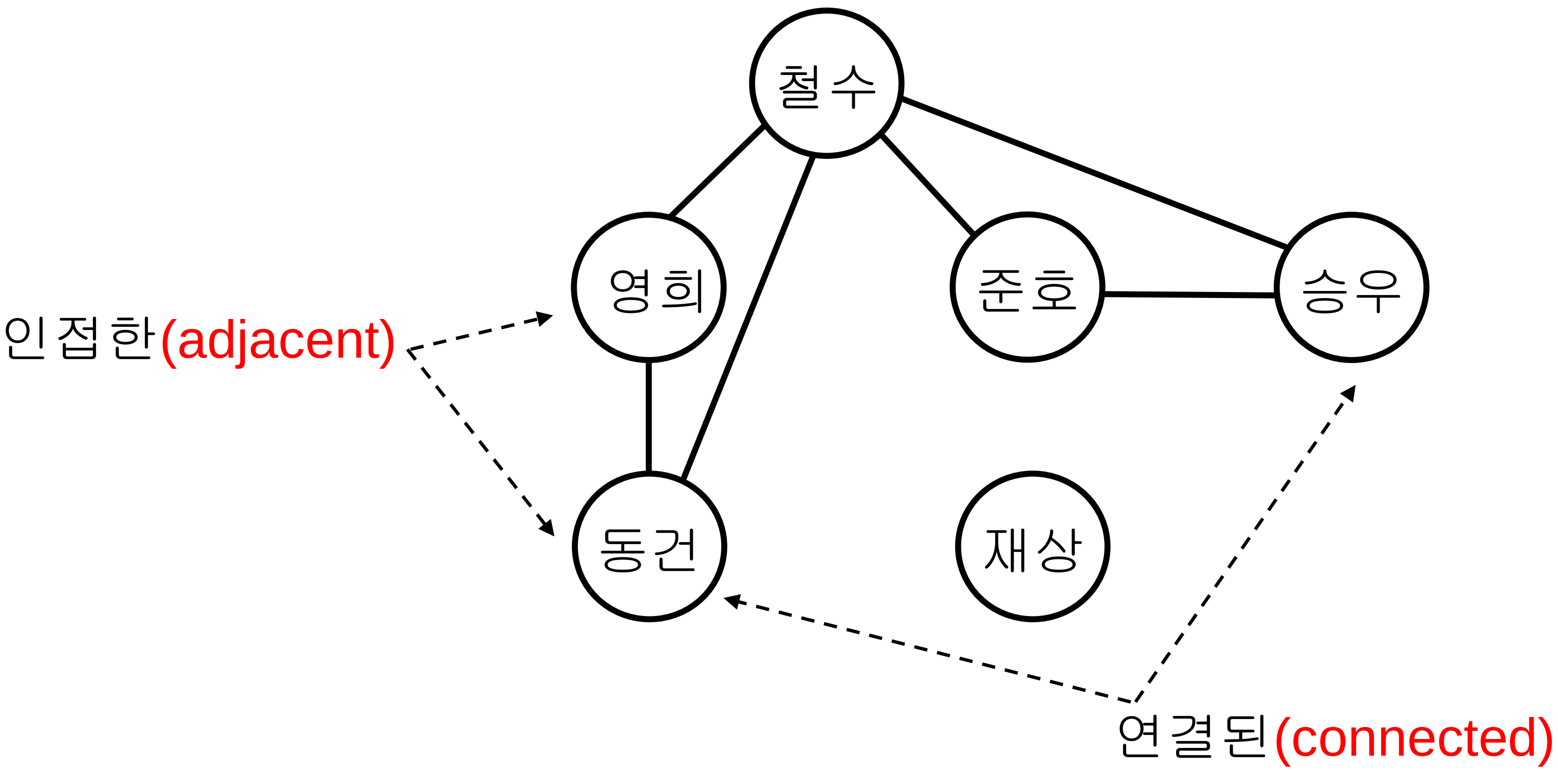


친분관계 그래프 – 방향과 가중치를 고려한 경우



그래프 용어

- 두 정점 사이에 간선이 존재하면 “인접하다”고 하며, 두 정점 사이에 경로가 존재하면 “연결되었다”고 함



그래프 용어

- 그래프의 종류
  - 무향 그래프(undirected graph) : 간선에 방향이 없음
  - 유향 그래프(directed graph; digraph) : 간선에 방향이 있음
  - 가중치 그래프(weighted graph) : 간선에 가중치가 있음
  - 연결 그래프(connected graph) : 모든 정점 간에 경로가 존재

학습내용

1. 그래프
2. **그래프의 표현**
3. 너비 우선 탐색(BFS)과 깊이 우선 탐색(DFS)
4. 최소 신장 트리
5. 위상 정렬
6. 최단 경로
7. 강연결 요소

그래프의 표현

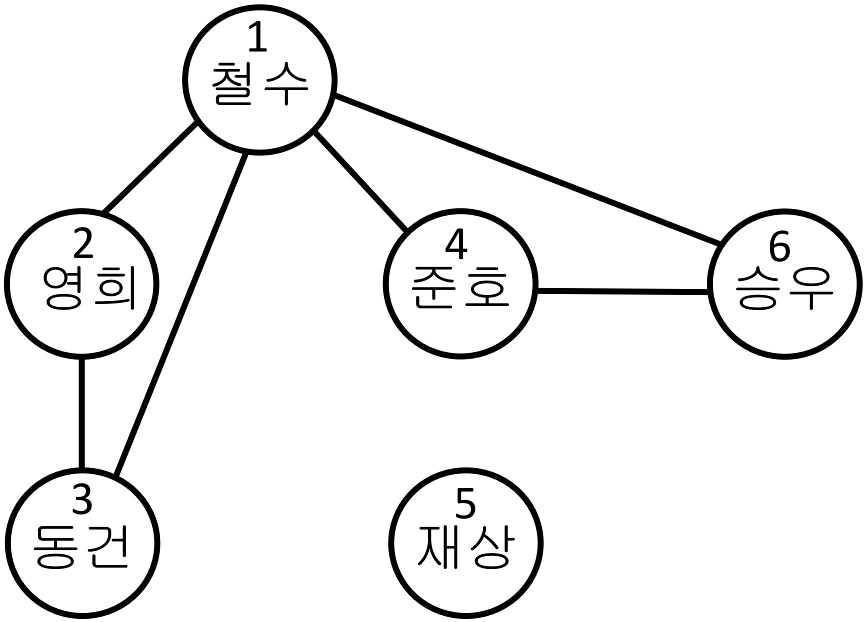
- 인접 행렬 방식
- 인접 리스트 방식
- 인접 배열 방식
- 인접 해시테이블 방식

Graph의 표현 - 인접 행렬

- Adjacency matrix
  - N x N 행렬로 표현

    N : 정점 수
    - 원소 $(i, j) = 1$: 정점 i 와 j 사이에 간선이 있음
    - 원소 $(i, j) = 0$: 정점 i 와 j 사이에 간선이 없음
  - directed graph의 경우
    - 원소 (i, j)는 정점 i 에서 정점 j 로 향하는 간선이 있는지를 나타냄
  - undirected graph의 경우
    - 원소 (i, j) = 원소 (j, i)
  - 가중치를 가진 그래프의 경우
    - 간선이 있으면 원소 (i, j)에 1 대신 가중치를 저장
    - 간선이 없으면 원소 (i, j)에 0을 저장(∞를 저장하는 경우도 있음)

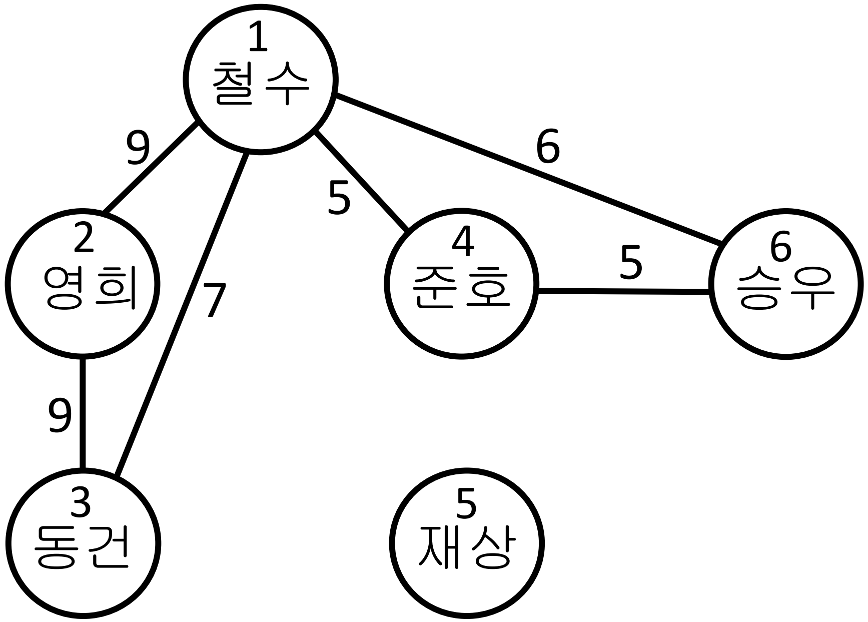
무향 그래프의 예



	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	0	0
4	1	0	0	0	0	1
5	0	0	0	0	0	0
6	1	0	0	1	0	0

1의 개수 = 간선 수 x 2

가중치 있는 무향 그래프의 예

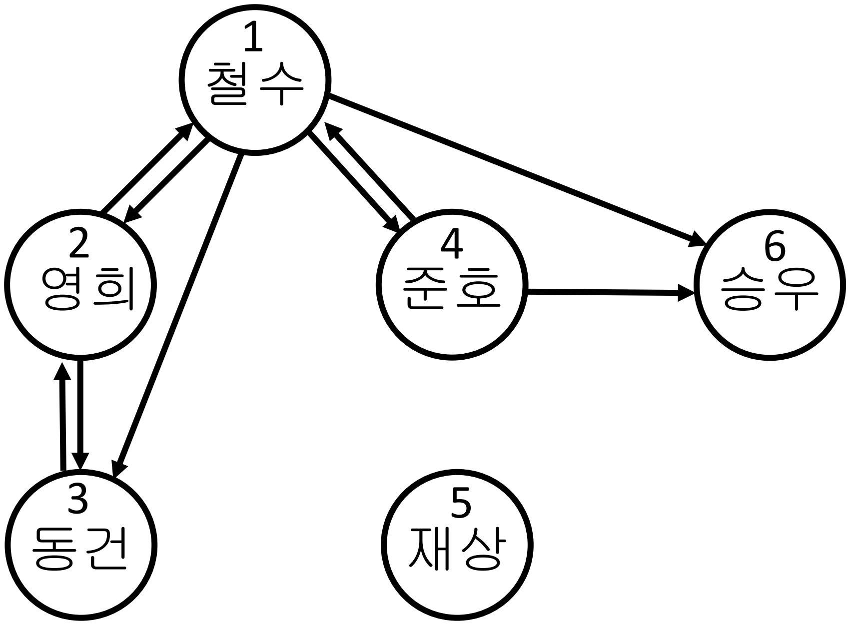


	1	2	3	4	5	6
1	0	9	7	5	0	6
2	9	0	9	0	0	0
3	7	9	0	0	0	0
4	5	0	0	0	0	5
5	0	0	0	0	0	0
6	6	0	0	5	0	0

0이 아닌 원소수 = 간선 수 x 2

간선이 없으면 가중치에 0을 저장

유향 그래프의 예



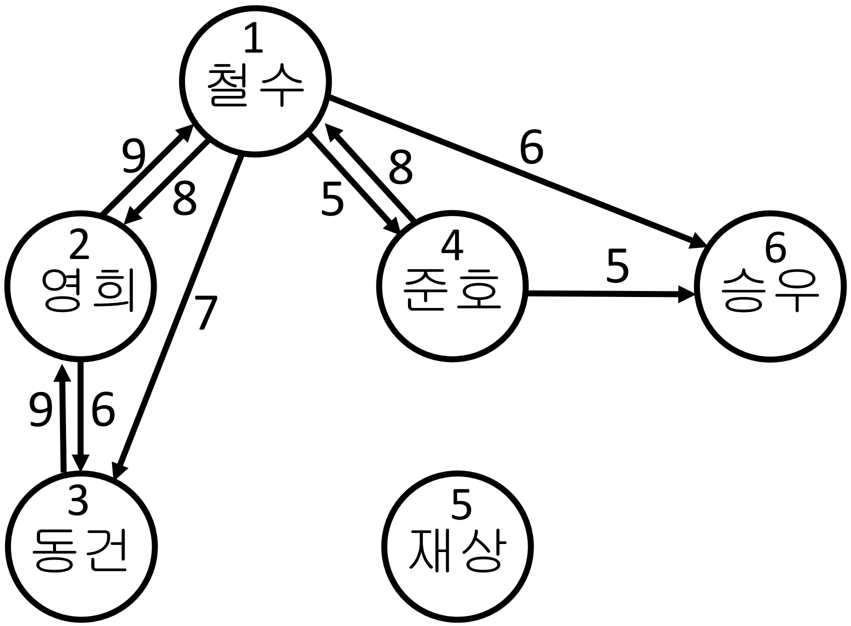
	1	2	3	4	5	6
1	0	1	1	1	0	1
2	1	0	1	0	0	0
3	0	1	0	0	0	0
4	1	0	0	0	0	1
5	0	0	0	0	0	0
6	0	0	0	0	0	0

1의 개수 = 간선 수

정점 1의 진출간선 수 :
정점 1의 out-degree =

정점 1의 진입간선 수 :
정점 1의 in-degree =

가중치 있는 유향 그래프의 예

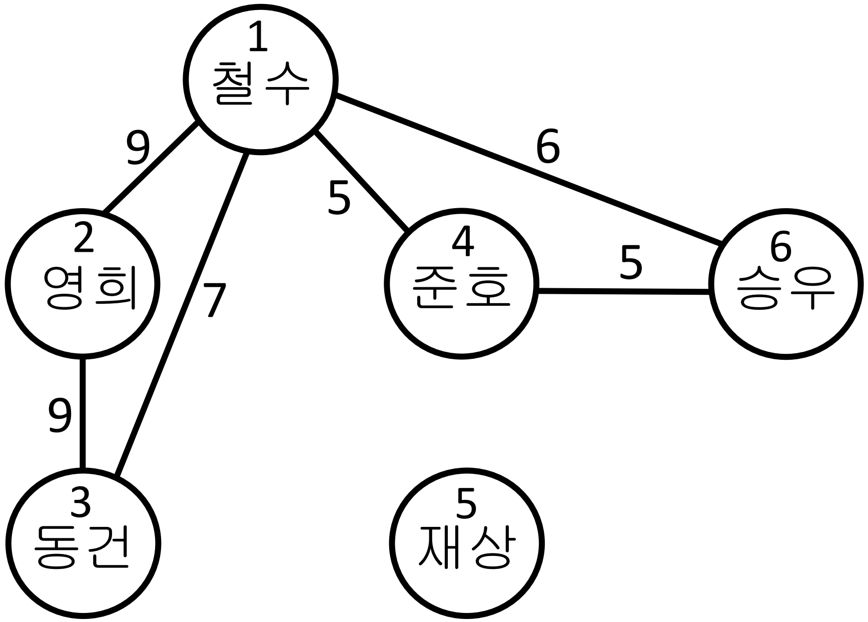


	1	2	3	4	5	6
1	0	8	7	5	0	6
2	9	0	6	0	0	0
3	0	9	0	0	0	0
4	8	0	0	0	0	5
5	0	0	0	0	0	0
6	0	0	0	0	0	0

0이 아닌 원소수 = 간선 수

간선이 없으면 가중치에 0을 저장

가중치 있는 그래프 표현의 다른 예



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

간선이 존재하면 가중치를 저장
간선이 없으면 가중치에 무한대 값을 저장
(앞의 예에서는 간선이 없으면 0을 저장했음)

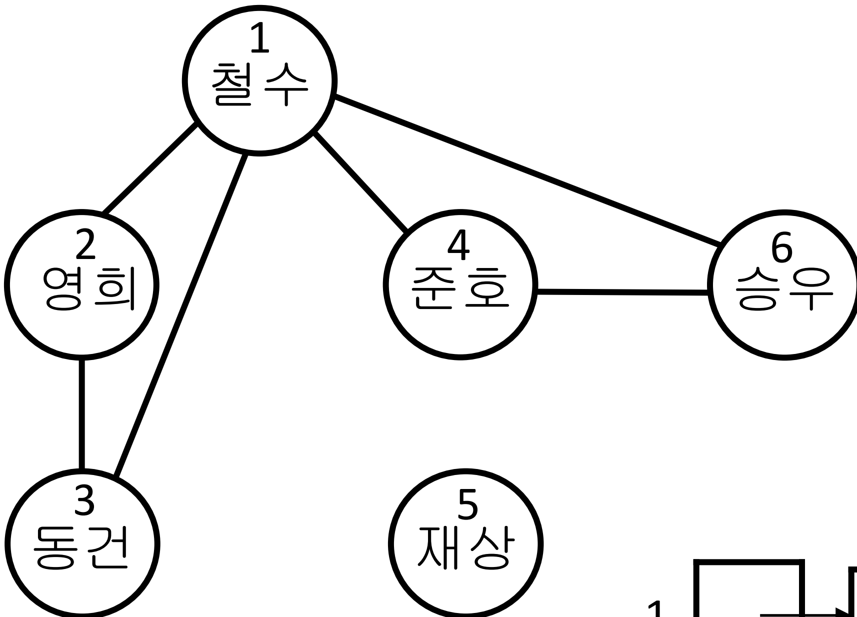
Graph의 표현 - 인접 리스트

- Adjacency list

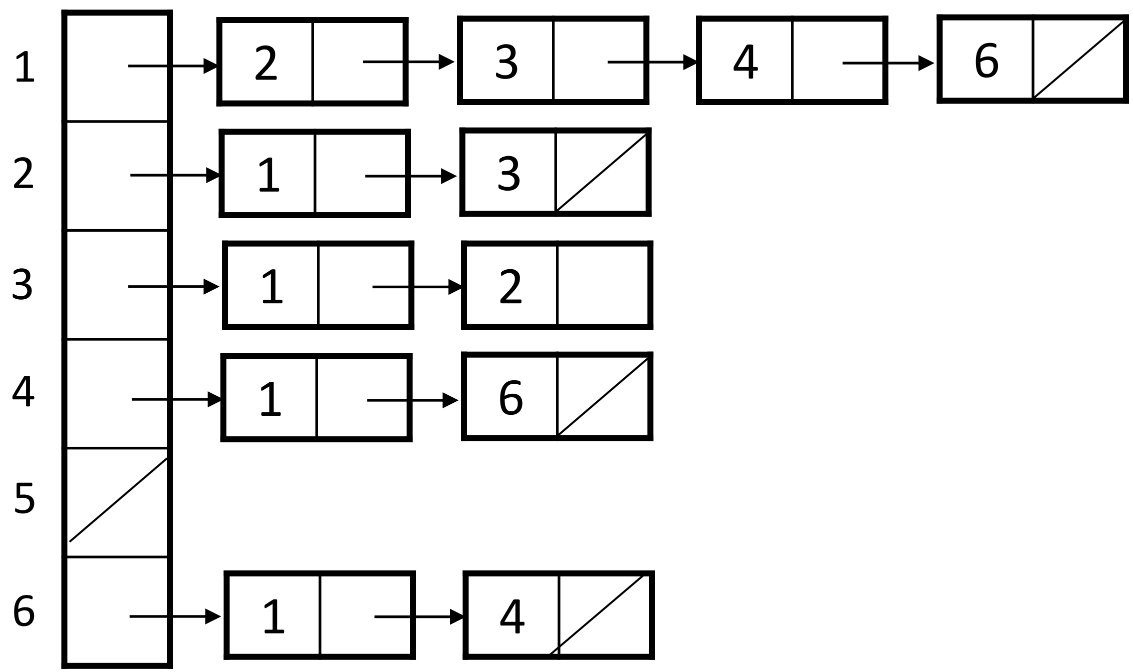
N : 정점 수

  - N 개의 연결 리스트로 표현
  - i 번째 리스트는 정점 i에 인접한 정점들을 모아 놓은 연결 리스트
  - 가중치 있는 그래프의 경우
    - 리스트에 정점 번호와 함께 가중치도 저장

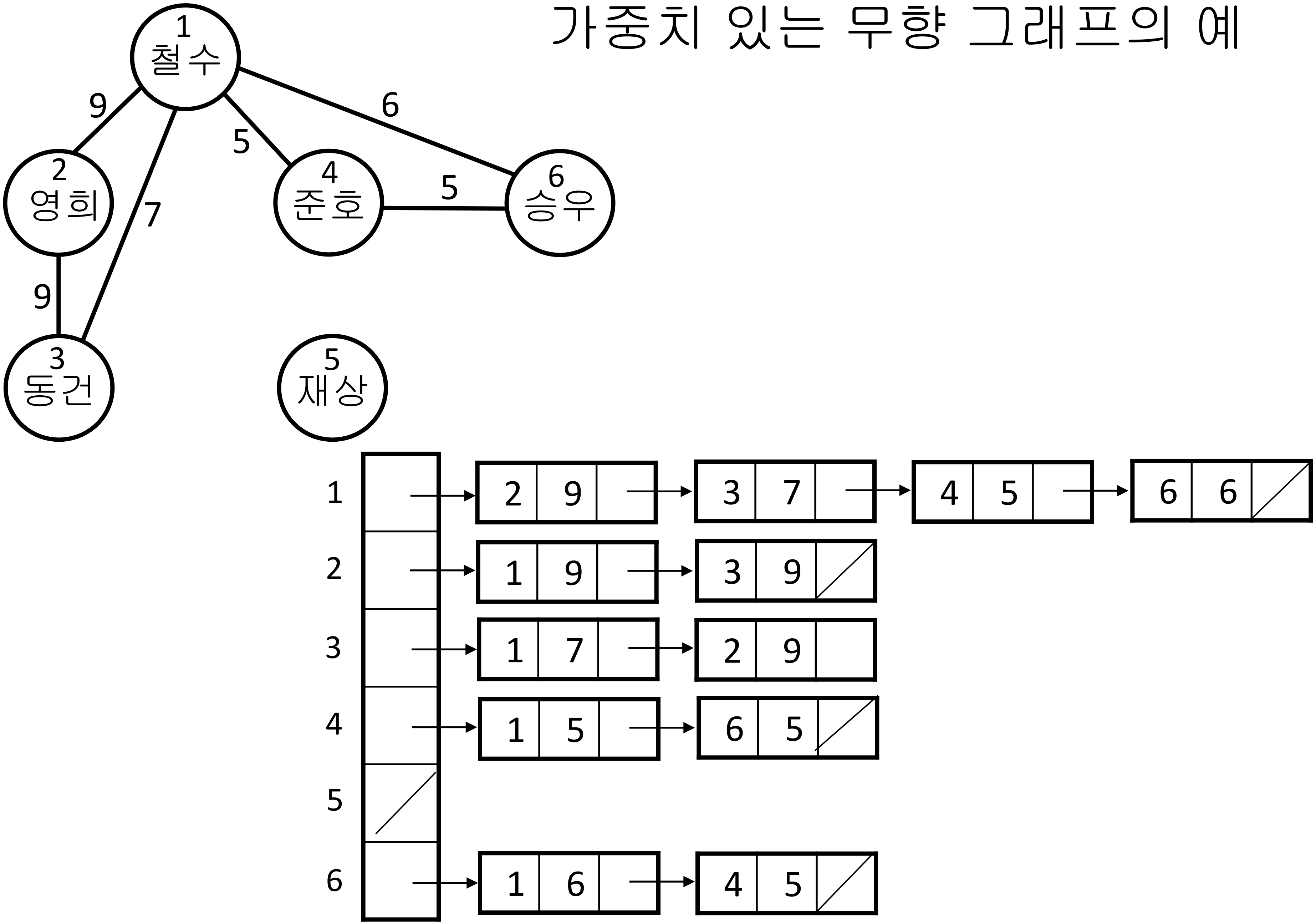
무향 그래프의 예



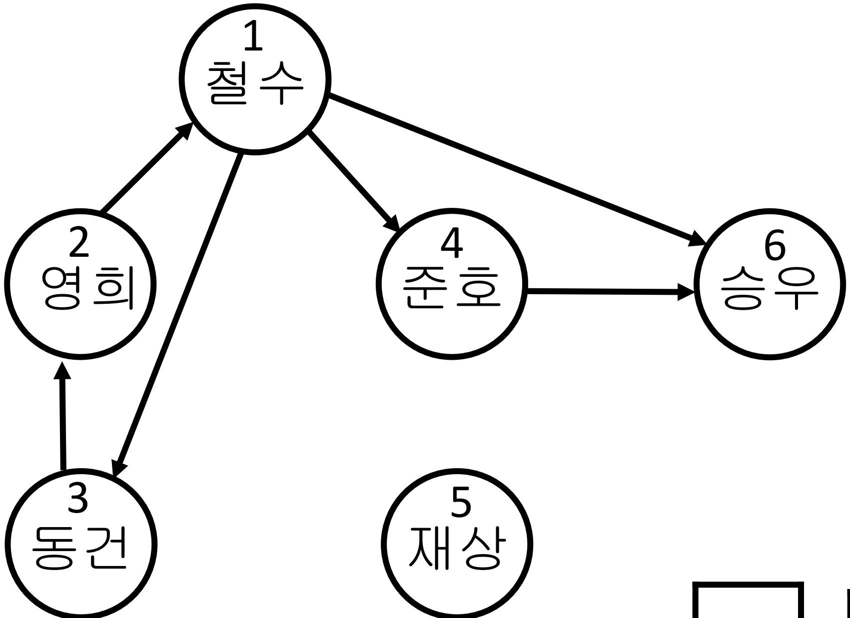
노드 수 = 간선 수 x 2



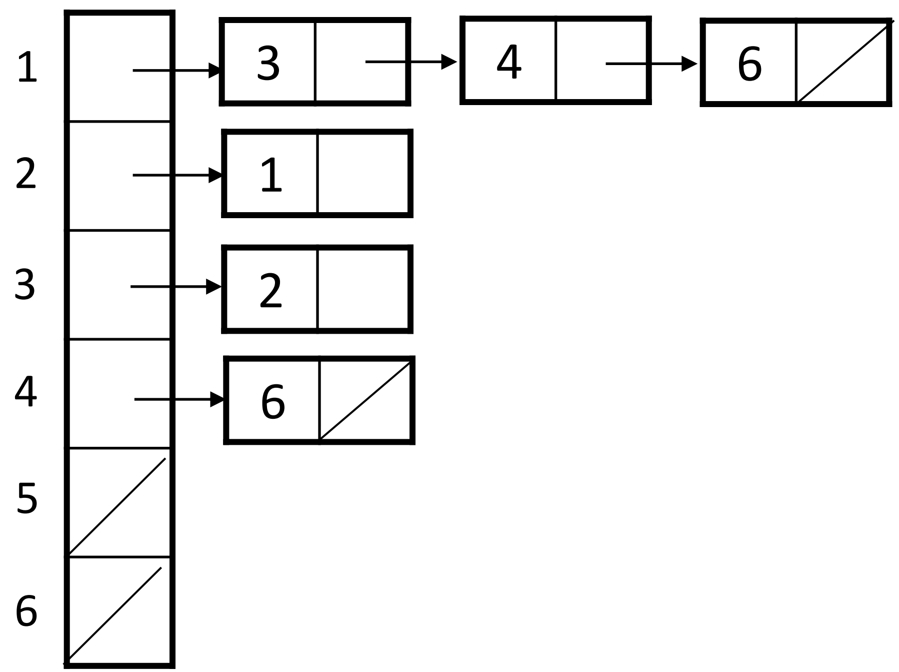
가중치 있는 무향 그래프의 예



유향 그래프의 예



노드 수 = 간선 수



Graph의 표현 - 인접 배열

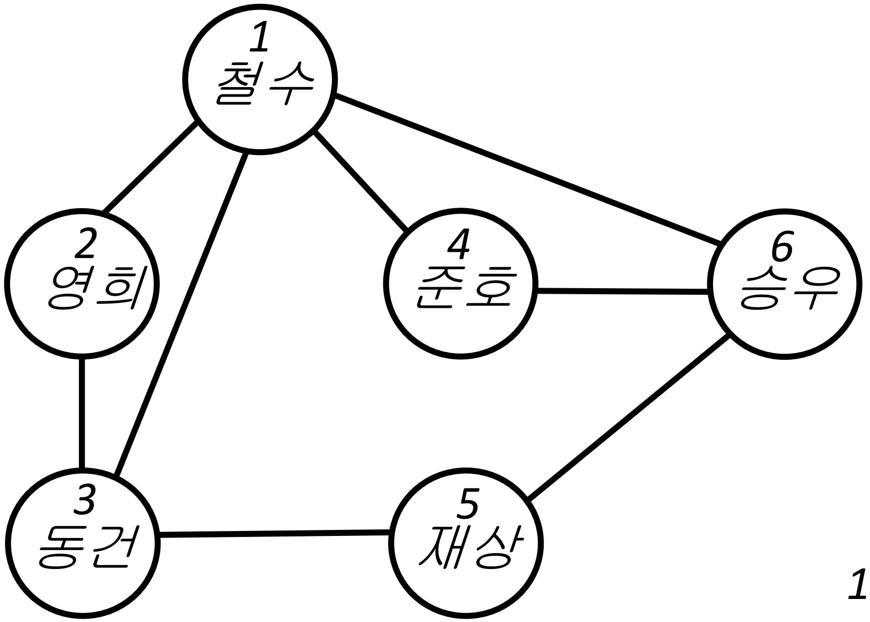
- 인접 행렬과 인접 리스트의 단점 중 하나
  - 인접 행렬 : 간선 수와 상관 없는 공간 차지
  - 인접 리스트 : 간선 (i, j) 존재 여부 검사 등의 연산은 시간이 오래 걸림

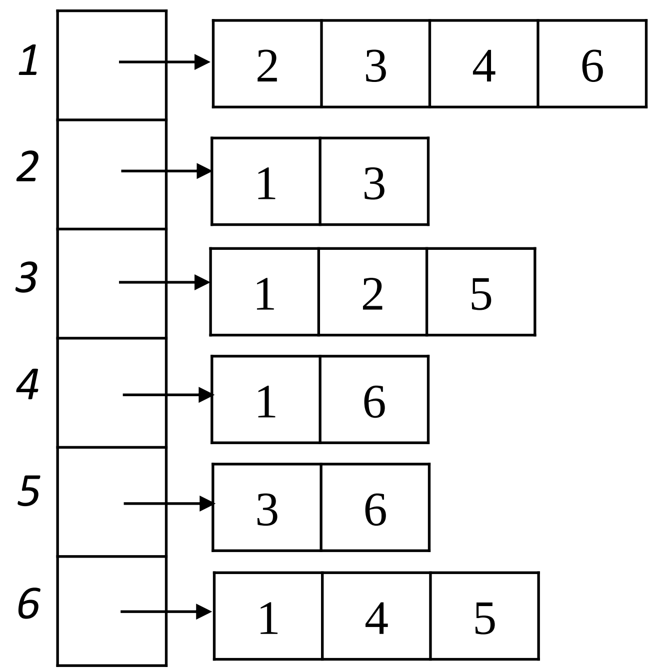
➔ 인접 배열 : 간선 수에 비례한 공간을 차지하면서 간선 존재 여부를 훨씬 빨리 검사할 수 있는 방법

인접 배열

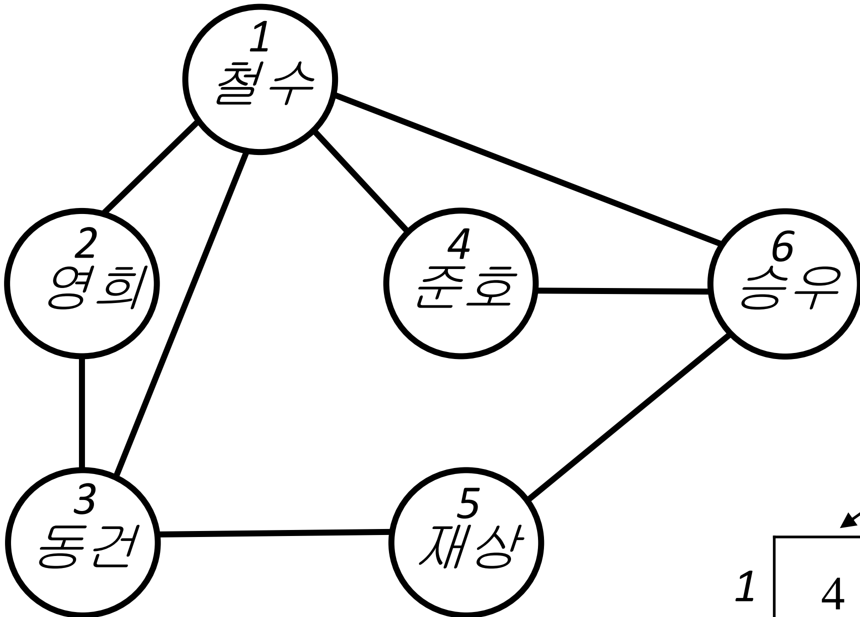
- 인접 배열을 이용한 그래프 표현
  - 정점이 N 개이면, N 개의 인접 배열로 표현
  - 각 정점에 인접한 정점들을 연결 리스트에 저장하는 대신, 배열에 저장. 즉, i 번째 배열은 정점 i 에 인접한 정점들의 집합
  - 각 배열을 정렬된 형태로 만들면 이진 탐색 가능
    - 정점 i의 인접 정점이 k 개인 경우, 이진 탐색을 이용하여 간선 (I, j) 존재 여부 검사하는 시간은 O(log k)

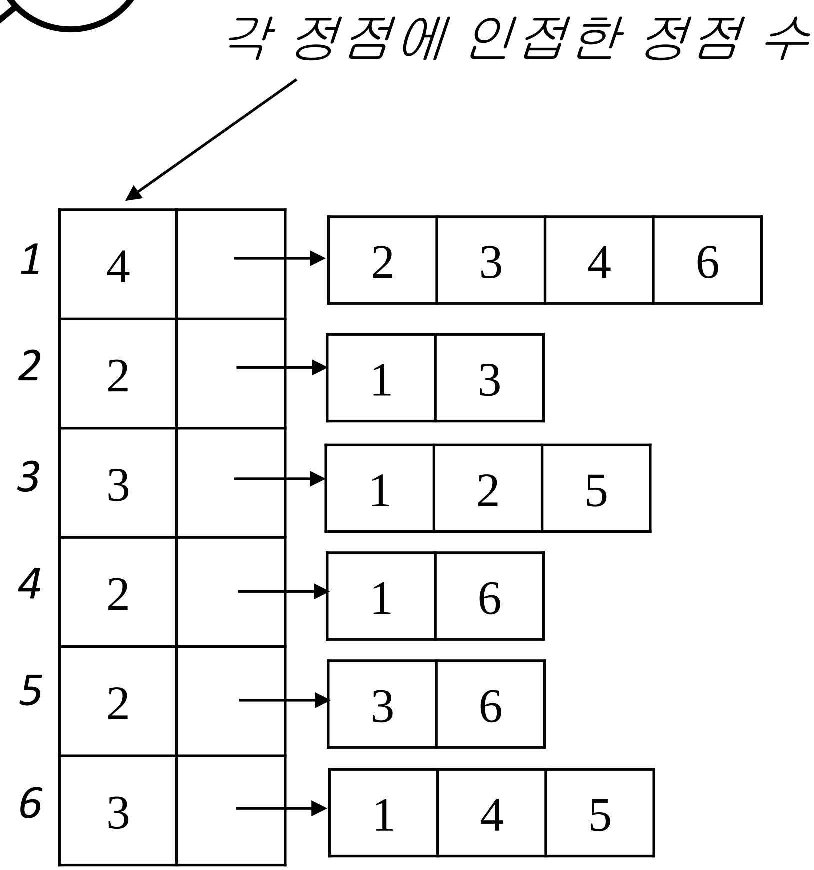
인접 배열



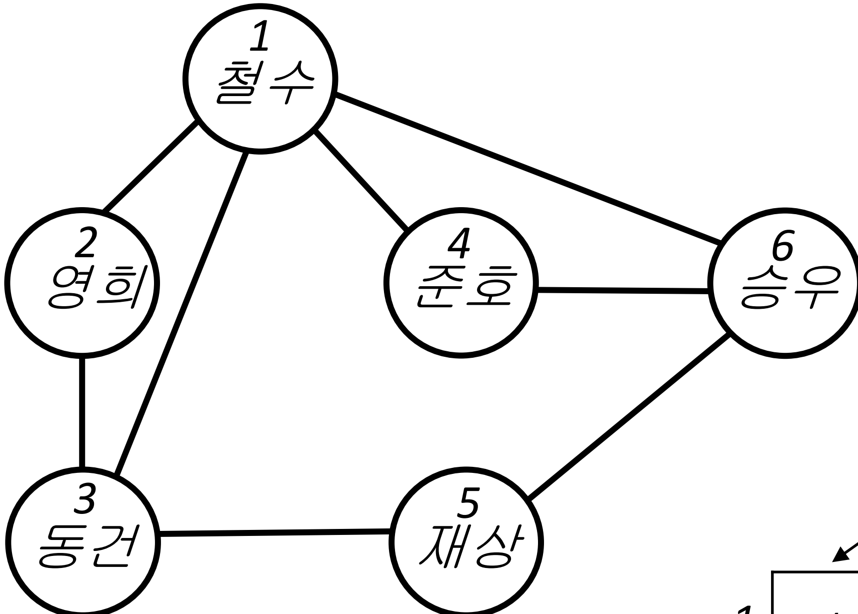


인접 배열

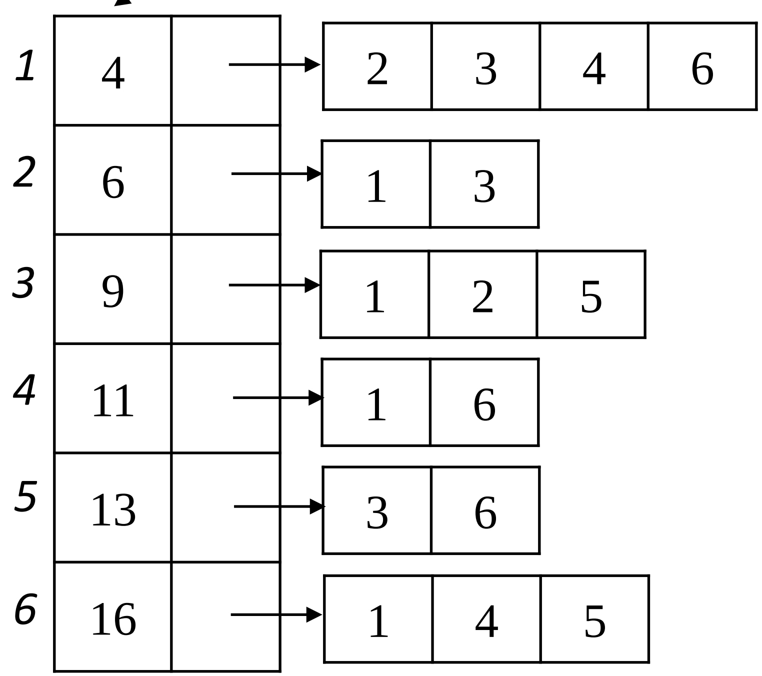

1 철수
2 영희
3 동건
4 준호
5 재상
6 승우


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

인접 배열



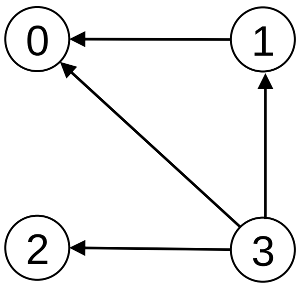
배열에서 각 정점에 인접한
정점 목록의 끝자리

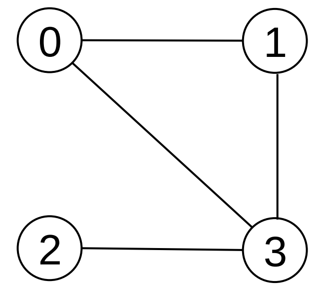


Graph의 표현 - 인접 해시테이블

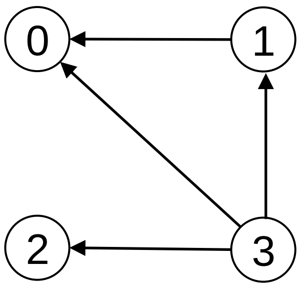
- 인접 해시테이블을 이용한 그래프 표현
  - 각 정점마다 하나의 인접 배열을 두는 대신에 하나의 해시 테이블을 사용
    - 정점이 N 개이면, N 개의 인접 해시 테이블로 표현
    - 각 정점마다 인접 배열의 2배 정도 공간을 할당 받아 해시 테이블의 적재율을 0.5 정도로 유지
  - 간선 (i, j) 존재 여부를 알아내는 연산은 효율적임
  - 정점 i에 인접한 정점들을 차례로 보면서 작업을 해야 할 경우는 적합하지 않음

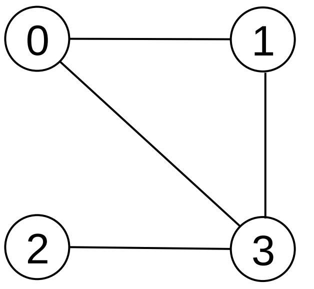
문제 : 다음 그래프를 인접 행렬로 표현하세요.



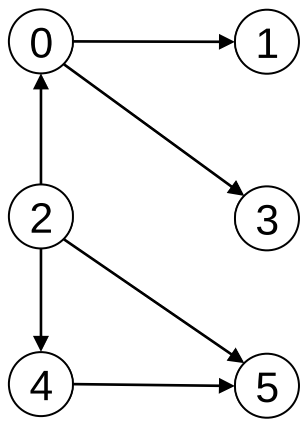


문제 : 다음 그래프를 인접 리스트로 표현하세요.

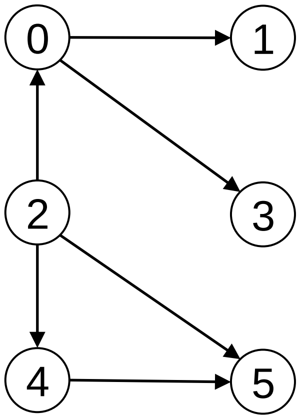




문제 : 다음 그래프를 인접 배열로 표현하세요.



문제 : 다음 그래프를 인접 해시 테이블로 표현하세요.



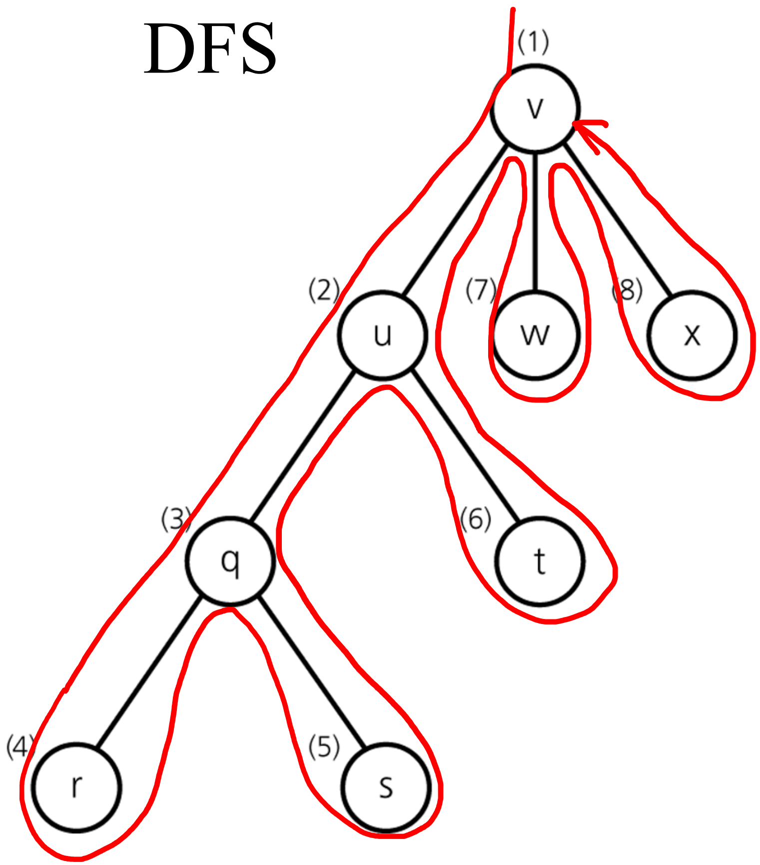
학습내용

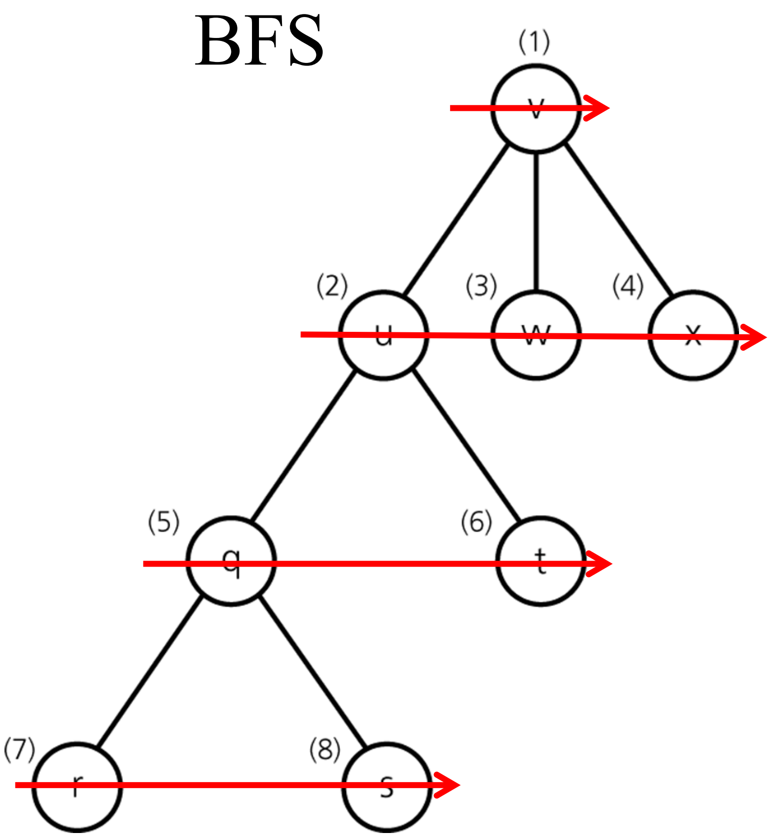
1. 그래프
2. 그래프의 표현
3. **너비 우선 탐색(BFS)과 깊이 우선 탐색(DFS)**
4. 최소 신장 트리
5. 위상 정렬
6. 최단 경로
7. 강연결 요소

Graph Traversal

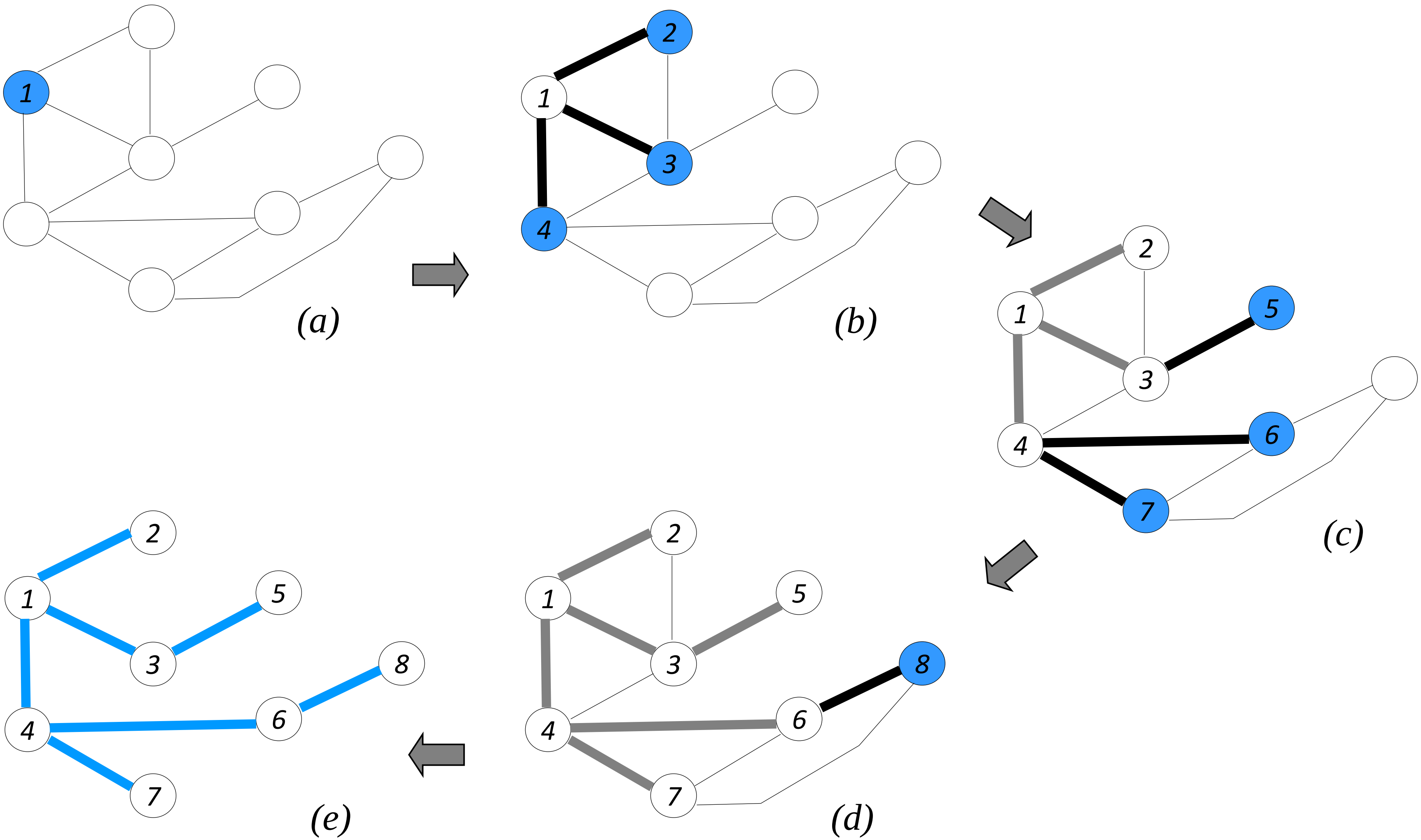
- 그래프의 모든 정점을 한번씩 방문
- 대표적 두 가지 방법
  - BFS (Breadth-First Search) : 너비 우선 탐색
    - 큐 이용하여 구현
  - DFS (Depth-First Search) : 깊이 우선 탐색
    - 스택 이용 또는 재귀 알고리즘으로 구현
- BFS와 DFS는 매우 중요한 알고리즘임
  - 그래프 알고리즘의 핵심
  - BFS와 DFS에 대해 매우 깊은 직관을 갖고 있어야 좋은 그래프 알고리즘을 만들 수 있음

동일한 그래프를 DFS/BFS로 탐색





BFS의 작동 예



너비우선트리

너비 우선 탐색 (BFS)

s에 연결된
정점만 방문

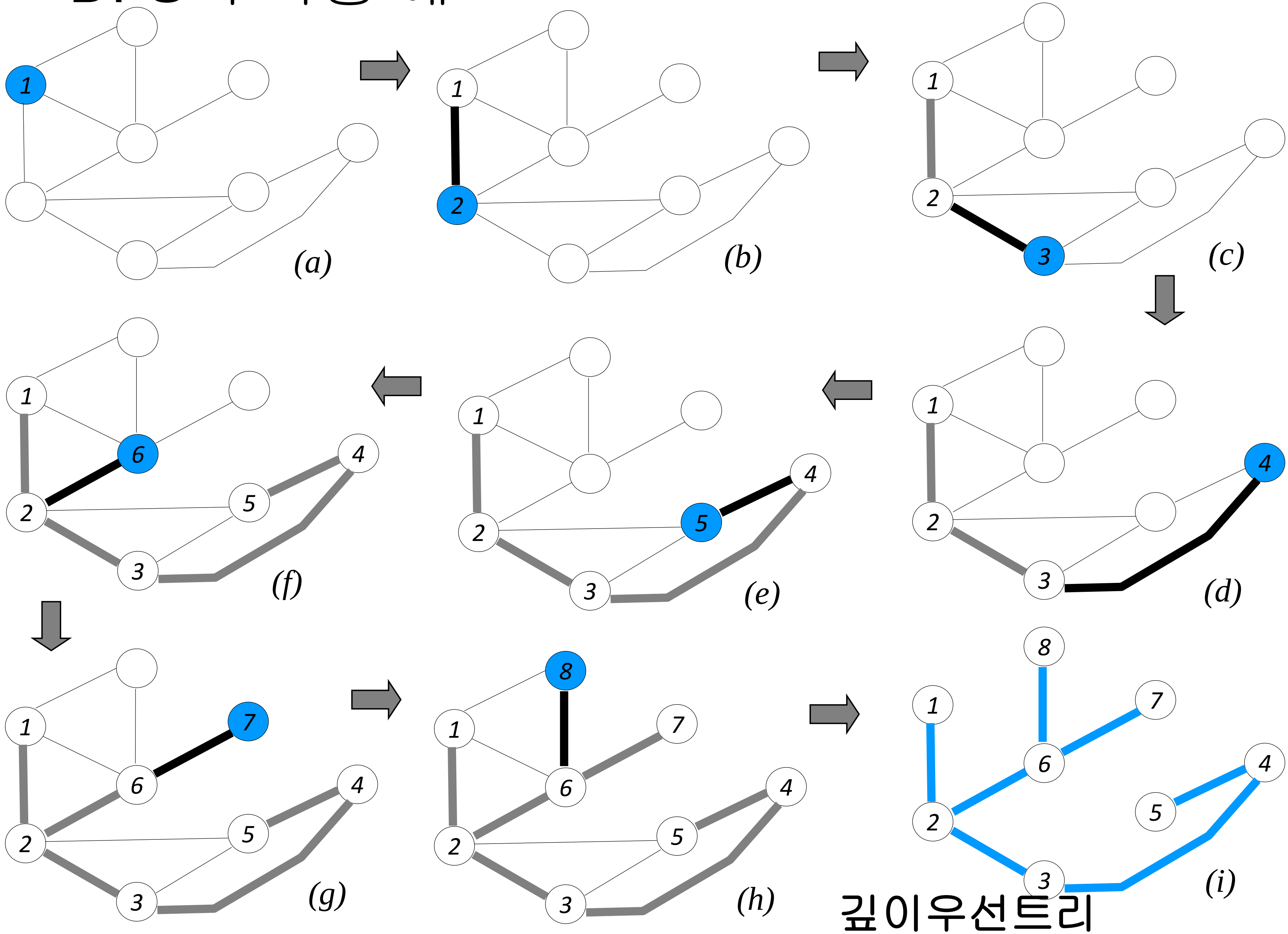
```
BFS(G, s)    ▷ s를 시작 정점으로 하여 그래프 G=(V, E)를 너비우선탐색
             ▷ 큐 Q를 이용
{
    for each v ∈ V(G)
        visited[v] ← NO;
    visited[s] ← YES;                    ▷ s: 시작 정점
    enqueue(Q, s);                       ▷ Q: 큐
    while (Q ≠ Φ) {
        v ← dequeue(Q);
        for each x ∈ L(v)                ▷ L(v) : 정점 v의 인접 정점 집합
            if (visited[x] = NO) then {
                visited[x] ← YES;
                enqueue(Q, x);
            }
    }
}
```

G가 연결 그래프
라고 가정

✓수행시간: $\Theta(|V|+|E|)$

✓혼동의 염려가 없으면 $\Theta(V+E)$로 표기해도 됨

DFS의 작동 예



깊이 우선 탐색 (DFS)

v에 연결된
정점만 방문

DFS(v) ▷ v를 시작 정점으로 하여 그래프를 깊이우선탐색
{
    visited[v] ← YES;
    for each $x \in L(v)$ ▷ $L(v)$: 정점 v의 인접 정점 집합
        if (visited[x] = NO) **then** DFS(x);
}

G가 연결 그래프
라고 가정

✓수행시간: $\Theta(|V|+|E|)$

깊이 우선 탐색 (DFS)

v에 연결된
정점만 방문

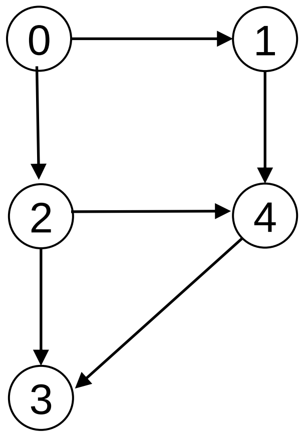
DFS(v) ▷ v를 시작 정점으로 하여 그래프를 깊이우선탐색
{
    visited[v] ← YES;
    for each $x \in L(v)$ ▷ $L(v)$: 정점 v의 인접 정점 집합
        if (visited[x] = NO) **then** DFS(x);
}

G가 연결그래프가
아니어도 모든
정점을 방문

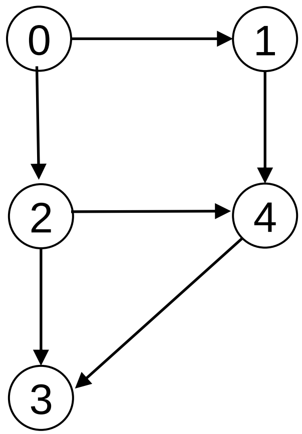
GraphTraversal(G) ▷ 그래프 G=(V, E)를 깊이우선탐색
{
    for each $v \in$ V(G)
        visited[v] ← NO;
    for each $v \in$ V(G)
        if (visited[v] = NO) **then** DFS(v);
}

✓수행시간: $\Theta(|V|+|E|)$

문제: 정점 0을 시작정점으로 한 BFS 순서를 적으세요.



문제: 정점 0을 시작정점으로 한 DFS 순서를 적으세요.



요약

- 그래프는 현상이나 사물을 정점(vertex)과 간선(edge)으로 표현하는 것으로서, 정점은 대상이나 개체를 나타내고 간선은 이들 간의 관계를 나타낸다.
- 그래프를 표현하는 방법은 인접행렬 방식, 인접 리스트 방식, 인접 배열 방식, 인접 해시테이블 방식 등이 있다.
- 대표적인 그래프 순회 방법은 너비 우선 탐색(BFS)과 깊이 우선 탐색(DFS)이다.

학습내용

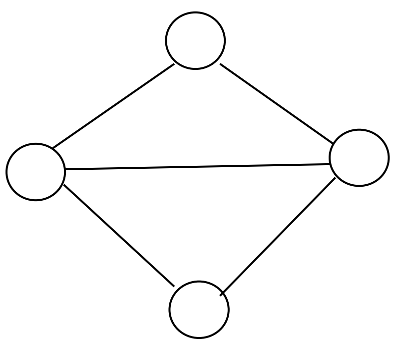
1. 그래프
2. 그래프의 표현
3. 너비 우선 탐색(BFS)와 깊이 우선 탐색(DFS)
4. **최소 신장 트리**
5. 위상 정렬
6. 최단 경로
7. 강연결 요소

최소 신장 트리

- 신장 트리와 최소 신장 트리
- Prim 알고리즘
- Kruskal 알고리즘

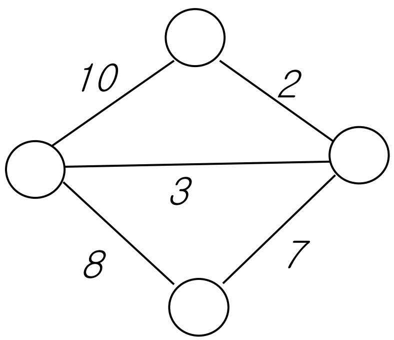
신장 트리(spanning tree)

- 조건
  - 무향 연결 그래프(undirected connected graph)
- 트리
  - 사이클(cycle)이 없는 연결 그래프
  - *n* 개의 정점을 가진 트리는 항상 *n*-1 개의 간선을 갖는다.
- 그래프 *G*의 신장 트리
  - *G*의 정점들과 간선들로만 구성된 트리(정점은 전부 포함)



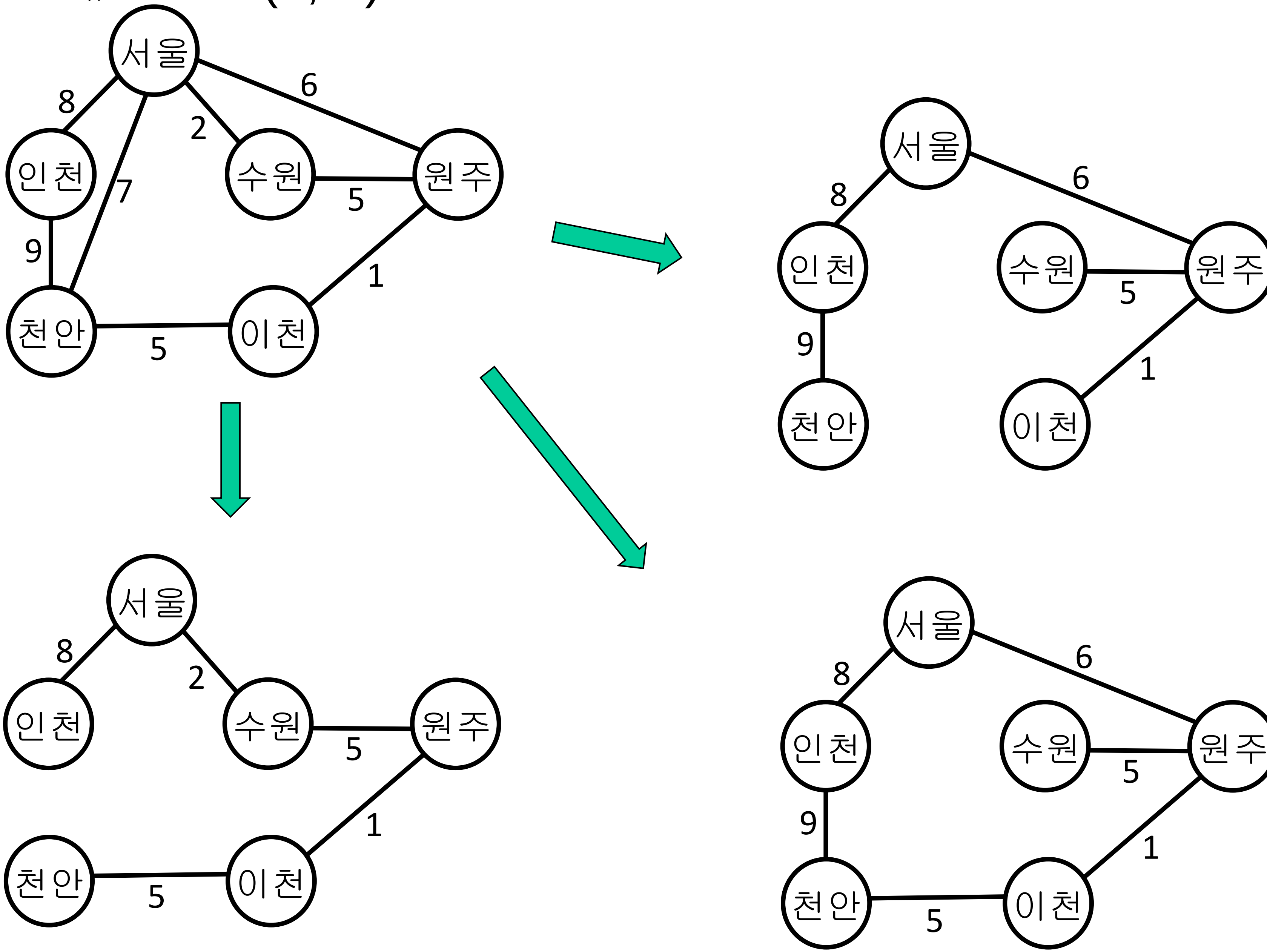
최소 신장 트리(minimum spanning tree)

- 그래프 *G*의 최소 신장 트리
  - 최소 비용 신장 트리
  - 무향 가중 연결 그래프 *G*의 신장 트리들 중 간선의 가중치 합이 최소인 것



- 그래프의 **최소 신장 트리**를 구하는 알고리즘을 알아보자.
  - Prim 알고리즘
  - Kruskal 알고리즘

그래프 G = (V, E)

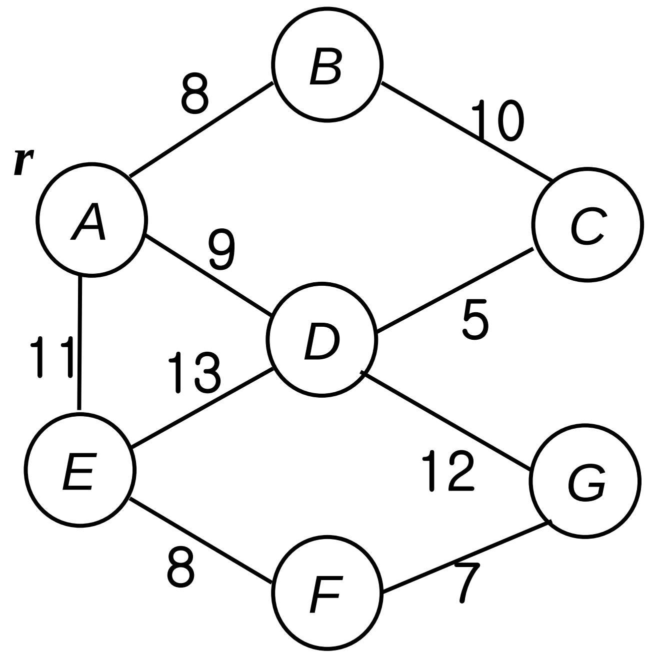


Prim 알고리즘

- 그래프 G=(V, E)의 최소 신장 트리를 이루는 정점 집합 S를 공집합에서 시작하여 그래프의 모든 정점을 포함할 때까지(S=V) 키워 나간다.
- 개념 설명

Prim(G, r) ▷ 정점 r로부터 시작하여 G=(V, E) 의 최소신장트리를 구함
{
    $S \leftarrow \Phi$; $T \leftarrow \Phi$; ▷ S: 정점 집합, T: 간선 집합
    정점 r을 집합 S에 포함시킴;
    while ($S \neq V$) {
        S에서 V-S를 연결하는 간선들 중 최소 길이인 (x, y) 를 찾음;
                                                    ▷ ($x \in S$, $y \in V$-S)
        간선 (x, y)를 T에 포함시킴;
        정점 y를 집합 S에 포함시킴;
    }
}

Prim 알고리즘 작동 예 – 개념 설명



Prim 알고리즘

Prim(G, r) ▷ 정점 r로부터 시작하여 G=(V, E) 의 최소신장트리를 구함

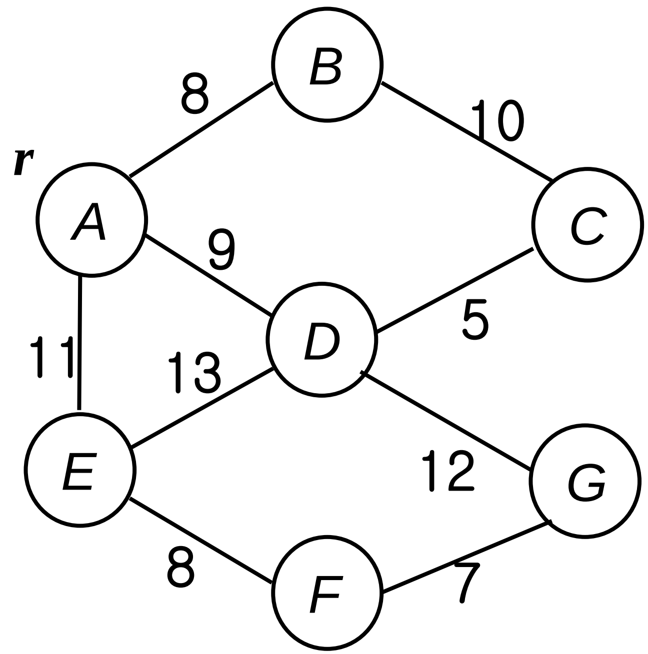
```
{
    Q ← V ;            ▷ Q : 트리 정점 집합 S에 속하지 않은 정점들의 집합
    for each u∈Q
        d[u] ← ∞;
    d[r] ← 0;
    while (Q ≠ Φ) {              ▷ |V| 번 반복
        u ← deleteMin(Q, d);
        for each v∈L(u)          ▷ L(u) : 정점 y의 인접 정점 집합
            if (v∈Q and d[v] > w(u, v)) then {
                d[v] ← w(u, v);
                tree[v] ← u;     ▷ u를 v의 부모로 등록(임시로)
            }
}
deleteMin(Q, d[])
{
    집합 Q에서 d 값이 가장 작은 정점 u를 리턴하고 u를 집합 Q에서 제거;
}
```

d[u] = 정점 u가 트리 정점에 인접하기 위한 간선 중 최소값. 즉, 신장 트리에 연결하는 현재까지 알려진 최소 비용

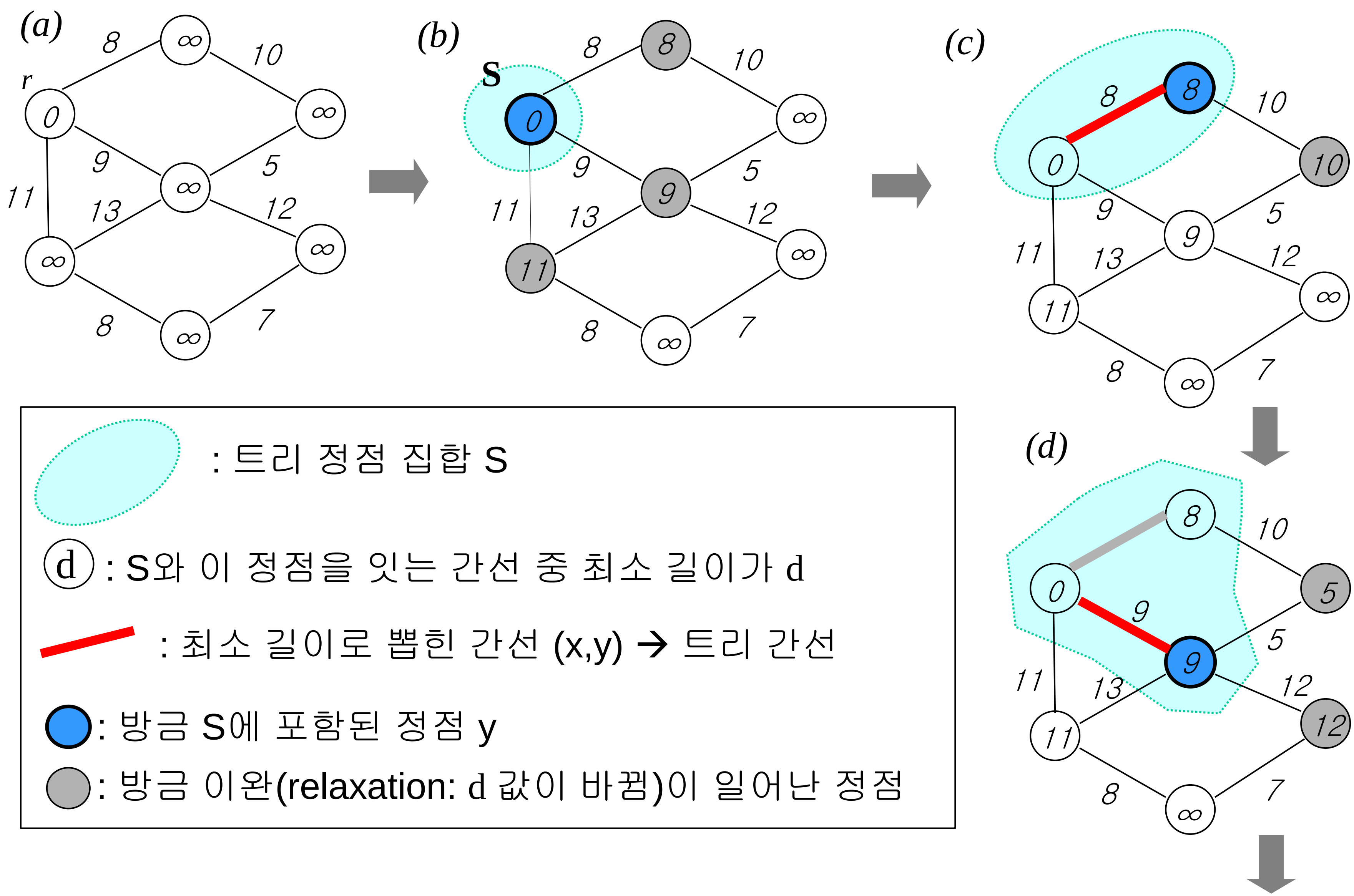
$w(u, v)$ = 간선 (u, v)의 가중치

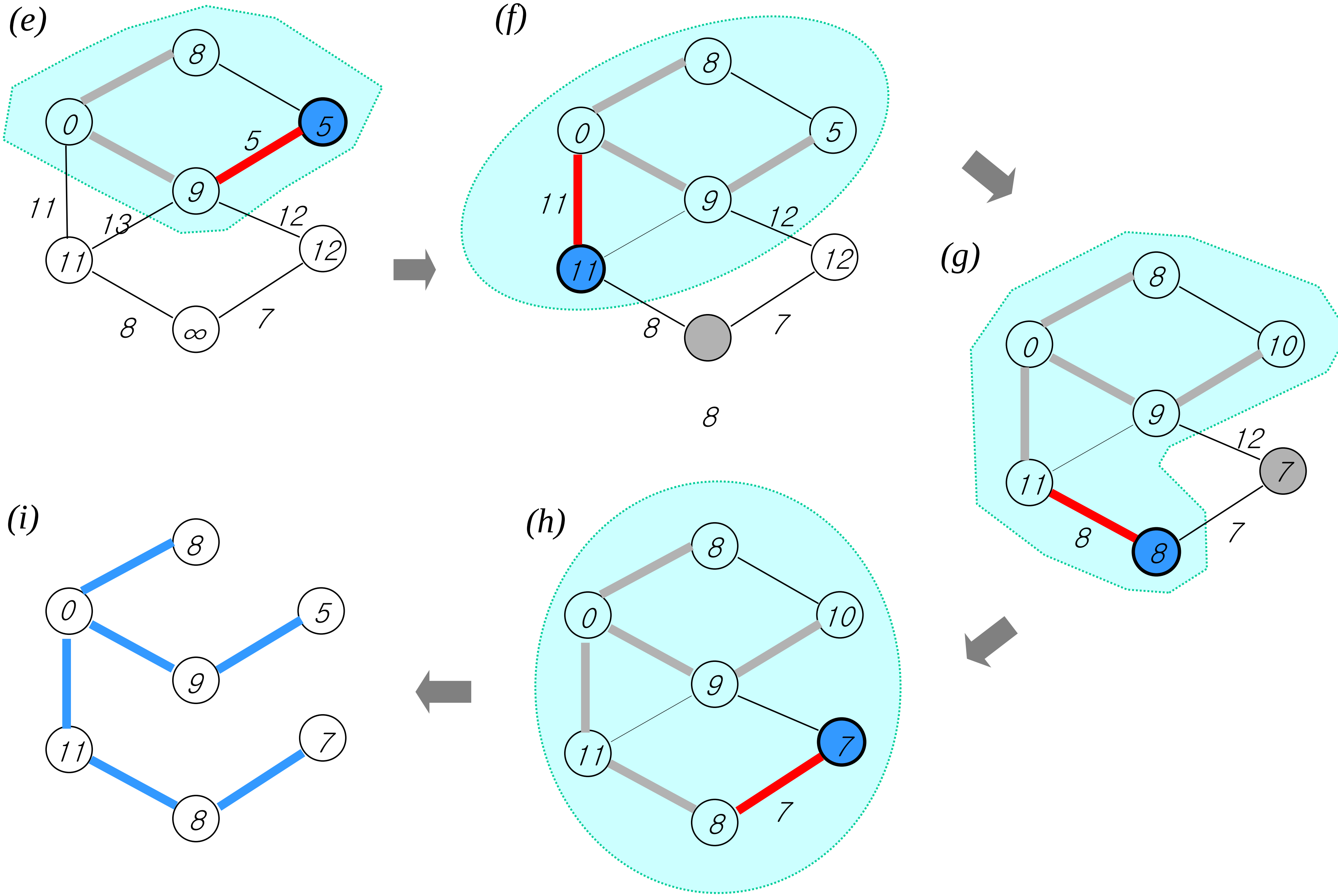
✓수행시간: $O(|E|log|V|)$ ← 최소 힙 이용시

Prim 알고리즘 작동 예 – 구현 설명



Prim 알고리즘의 작동 예





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

Kruskal 알고리즘

- 그래프 G=(V, E)의 최소 신장 트리를 이루는 간선 집합 T를 다음과 같은 방법으로 구함
  - T에 간선이 하나도 없는 상태에서 시작한다.
  - cycle을 만들지 않는 범위에서 최소 비용 간선을 하나씩 T에 추가해 나간다.
  - T에 간선이 |V|-1개 포함되면 완료한다.

Kruskal 알고리즘

$O((V+E)\log^* V)$

Kruskal(G) ▷ $G=(V, E)$ 의 최소신장트리 T를 구함
{
 $T \leftarrow \Phi$; ▷ T : 신장트리
 각각 하나의 정점만으로 이루어진 $|V|$개의 집합을 구성함;
 모든 간선을 가중치 크기 오름차순으로 정렬하여 배열 A에 저장;

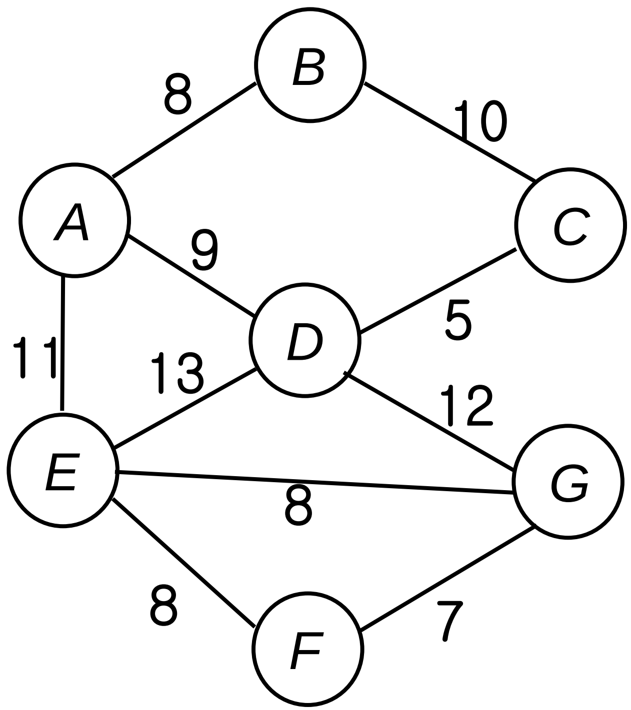
$O(E \log E) = O(E \log V)$

 while ($|T| < |V|-1$) {
  A에서 최소비용 간선 (u, v)를 제거;
  if (정점 u와 정점 v가 서로 다른 집합에 속함) **then** {
   $T \leftarrow T \cup \{(u, v)\}$;
   u와 v가 속한 두 집합을 하나로 합침;
  }
 }
}

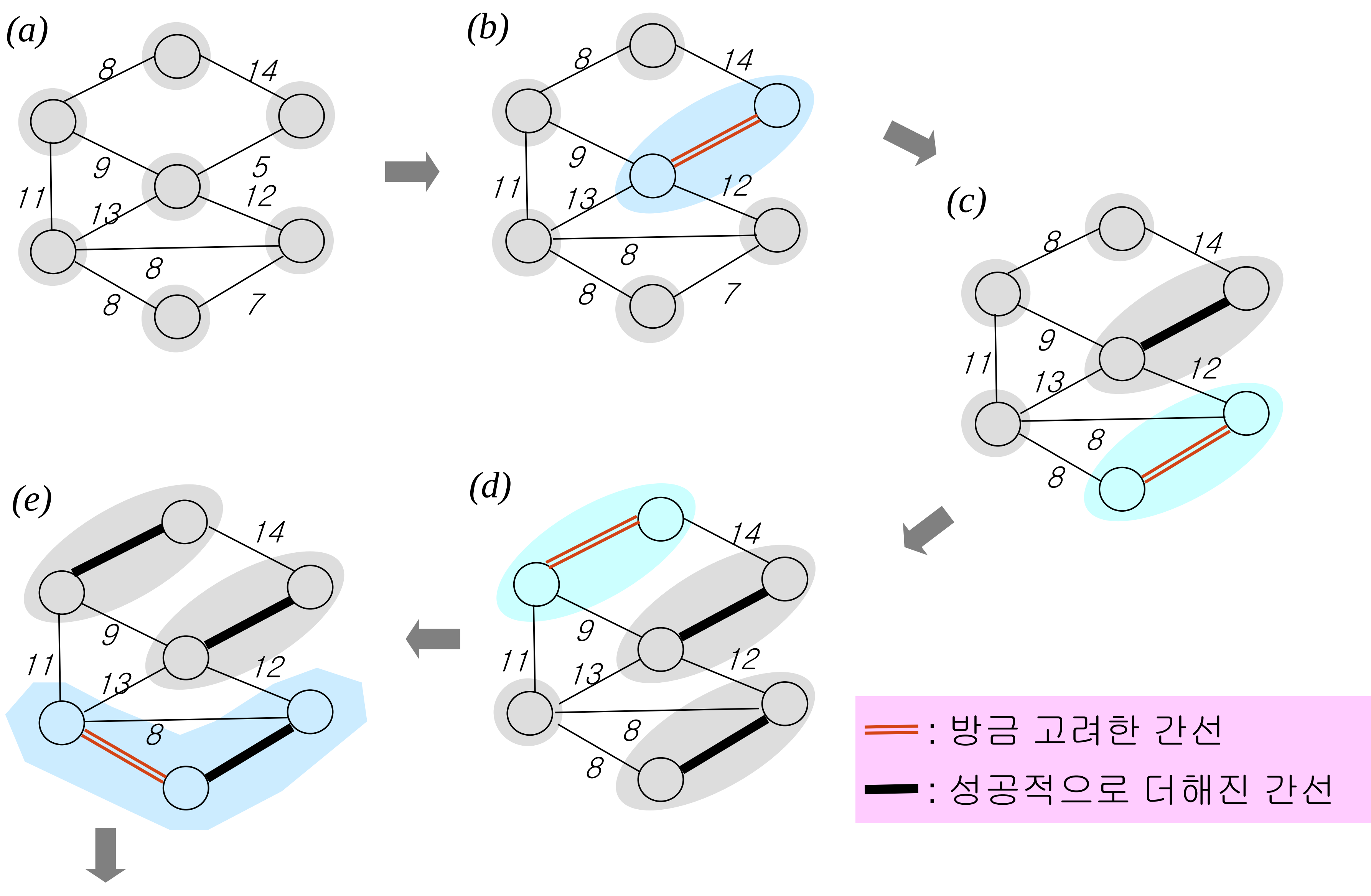
트리를 이용하여 집합 구현 -
랭크를 이용한 Union과
경로압축을 이용한 Find-Set

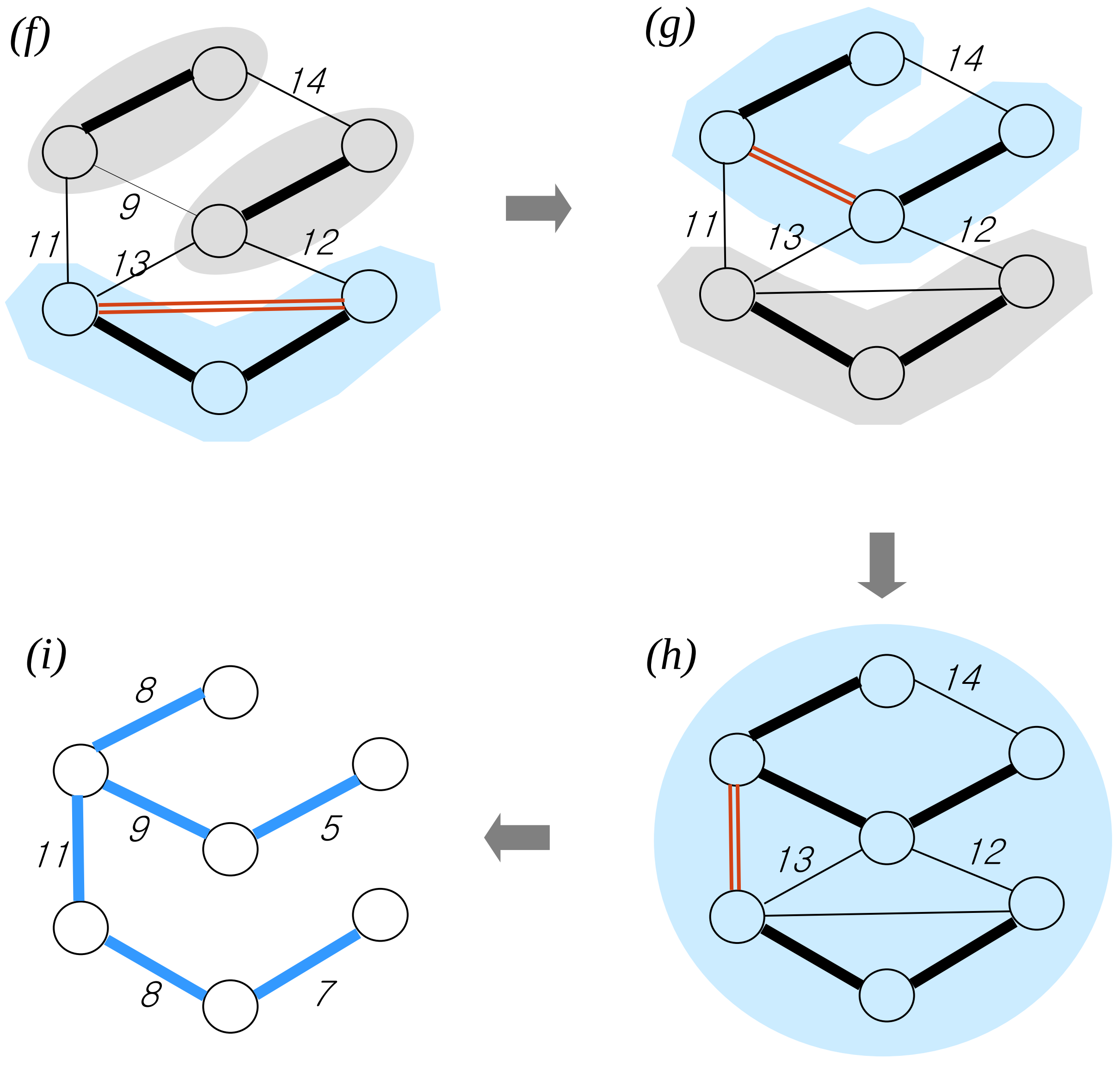
✓ Kruskal 알고리즘의 수행시간: $O(|E|log|V|)$

Kruskal 알고리즘의 작동 예

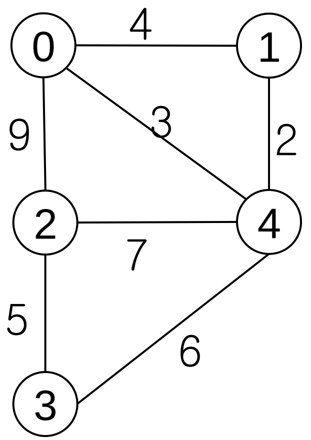


Kruskal 알고리즘의 작동 예

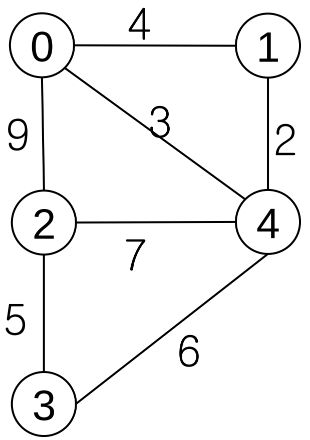



(f)
14
9
11
13
12
(g)
14
11
13
12
(h)
14
13
12
(i)
8
11
9
5
8
7

문제: Prim 알고리즘을 이용하여 최소 신장 트리를 구하세요.



문제: Kruskal 알고리즘을 이용하여 최소 신장 트리를 구하세요.



요약

- 최소 신장 트리(minimum spanning tree)는 주어진 무향 가중 연결 그래프 G=(V, E)에서 간선을 |V|-1 개만 남겨 만들어지는 트리 중 가중치 합이 최소인 트리이다.
- Prim 알고리즘, Kruskal 알고리즘을 이용하여 최소 신장 트리를 구할 수 있다.

학습내용

1. 그래프
2. 그래프의 표현
3. 너비 우선 탐색(BFS)와 깊이 우선 탐색(DFS)
4. 최소 신장 트리
5. **위상 정렬**
6. 최단 경로
7. 강연결 요소

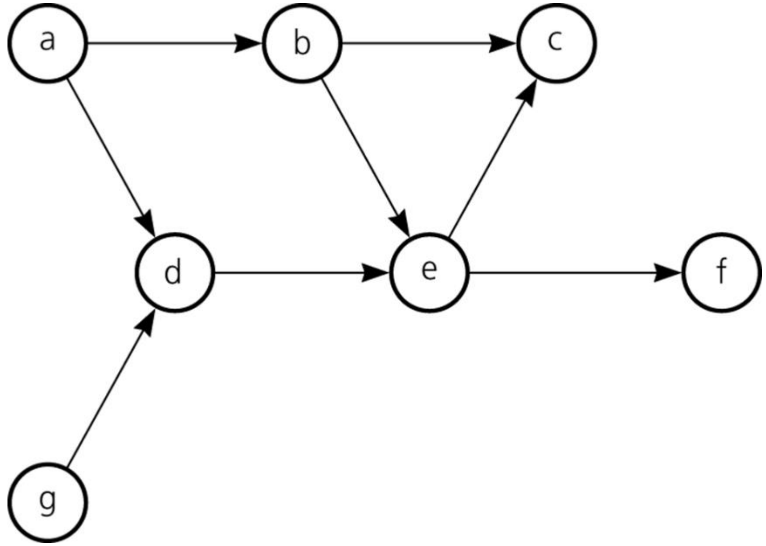
위상 정렬(Topological Sort)

- DAG의 위상 정렬
- 위상 정렬 알고리즘 1
- 위상 정렬 알고리즘 2
  - 알고리즘 1보다 많이 사용되는 방법
  - DFS를 거의 그대로 이용하므로 구현이 간단

DAG의 위상 정렬

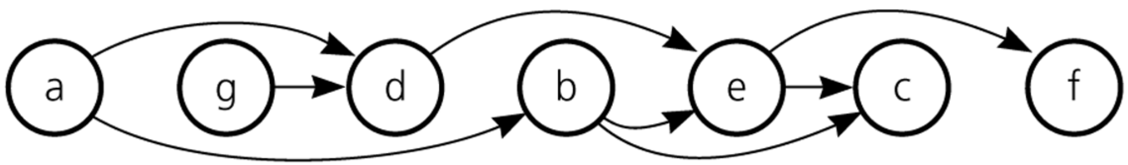
- 조건
  - cycle이 없는 유향 그래프
    (DAG: Directed Acyclic Graph)
- 위상 정렬(topological sort)
  - 모든 정점을 일렬로 나열하되
  - 정점 x에서 정점 y로 가는 간선이 있으면 x를 y보다 앞에 놓이도록 한다.
- 하나의 DAG에 하나 이상의 위상 순서가 존재
  - 즉, 위상 정렬 결과는 여러 가지일 수 있다.

G:

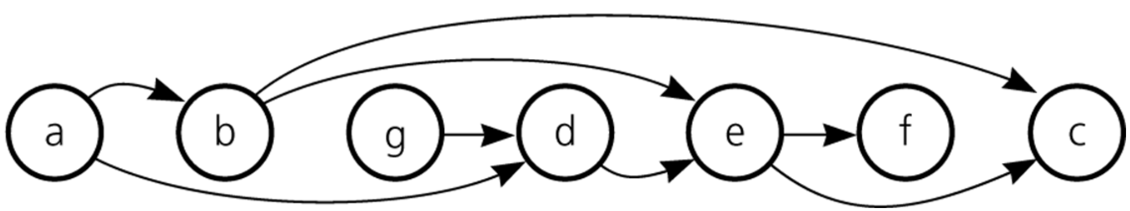


그래프 G의 위상정렬 예 2가지

(a)



(b)



위상정렬 알고리즘 1

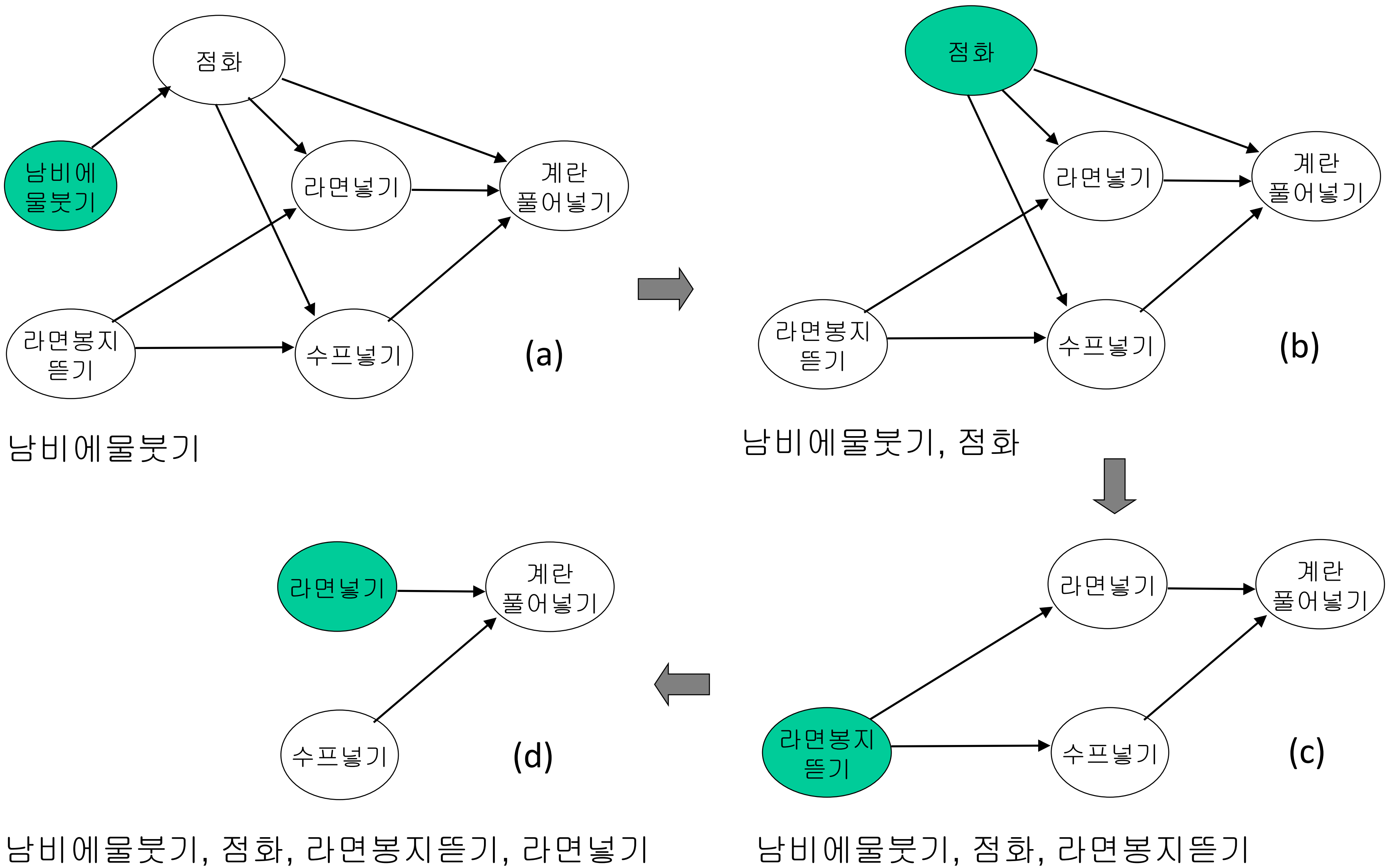
```
topological Sort1(G)   ▷ 그래프 G=(V, E)를 위상 정렬
{
      for i ← 1 to n {       ▷ n : 정점 수
            진입간선이 없는 정점 u를 선택한다;
            A[i] ← u;
            정점 u와, u의 모든 진출간선을 제거한다;
      }
}
```

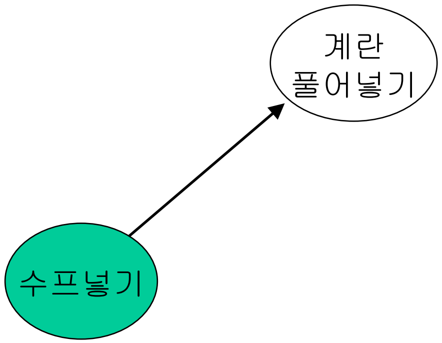
✓ 알고리즘 수행이 끝나고 나면 배열 A[1…n]에 정점들이 위상정렬 되어 있다.

✓수행시간: $\Theta(|V|+|E|)$

진입간선이 없는 정점 u를 선택하는 작업을
상수 시간에 수행할 수 있도록 구현한다고 가정

위상정렬 알고리즘 1의 작동 예





(e)

남비에물붓기, 점화, 라면봉지뜯기, 라면넣기, 수프넣기

(f)

남비에물붓기, 점화, 라면봉지뜯기, 라면넣기, 수프넣기, 계란풀어넣기

(g)

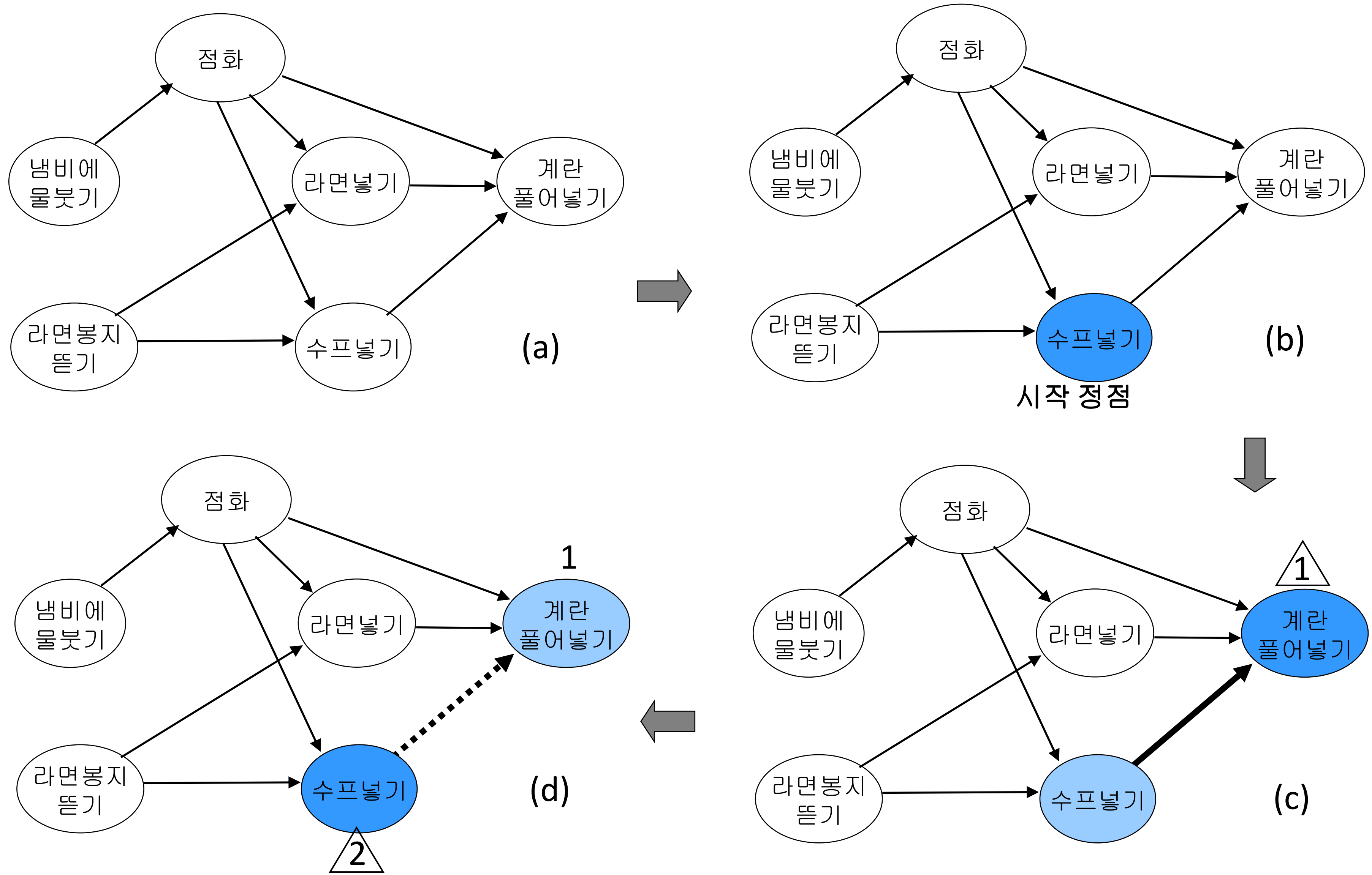
위상정렬 알고리즘 2

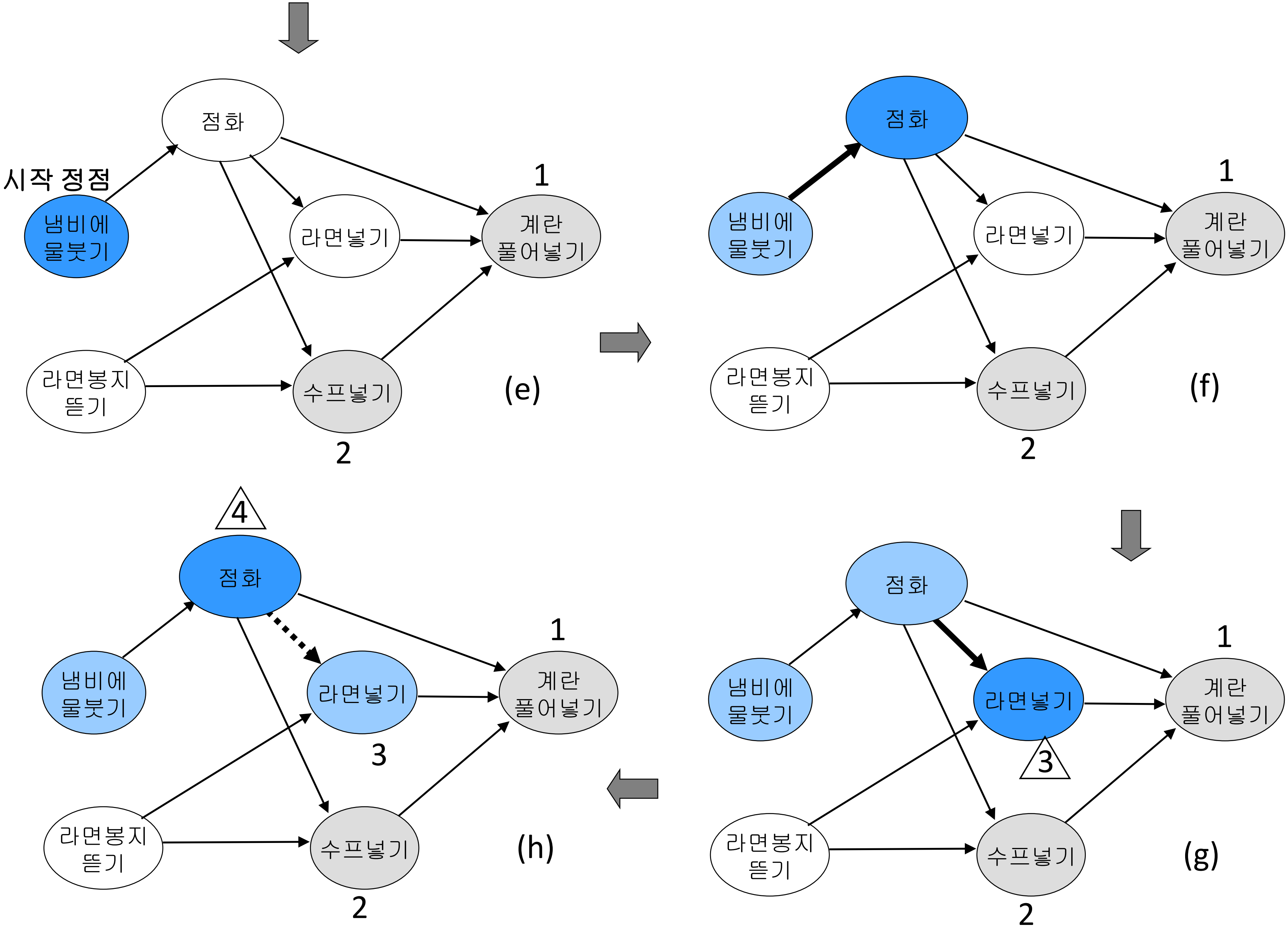
```
topologicalSort2(G) ▷ 그래프 G=(V, E)를 위상 정렬
{
      for each v∈V
            visited[v] ← NO;
      for each v∈V  ▷ 정점의 순서는 무관
            if (visited[v] = NO) then DFS-TS(v) ;
}

DFS-TS(v) ▷ v를 시작 정점으로 하여 그래프를 깊이우선탐색
{
      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 삽입;   ▷ DFS에 이 부분만 추가
}
```

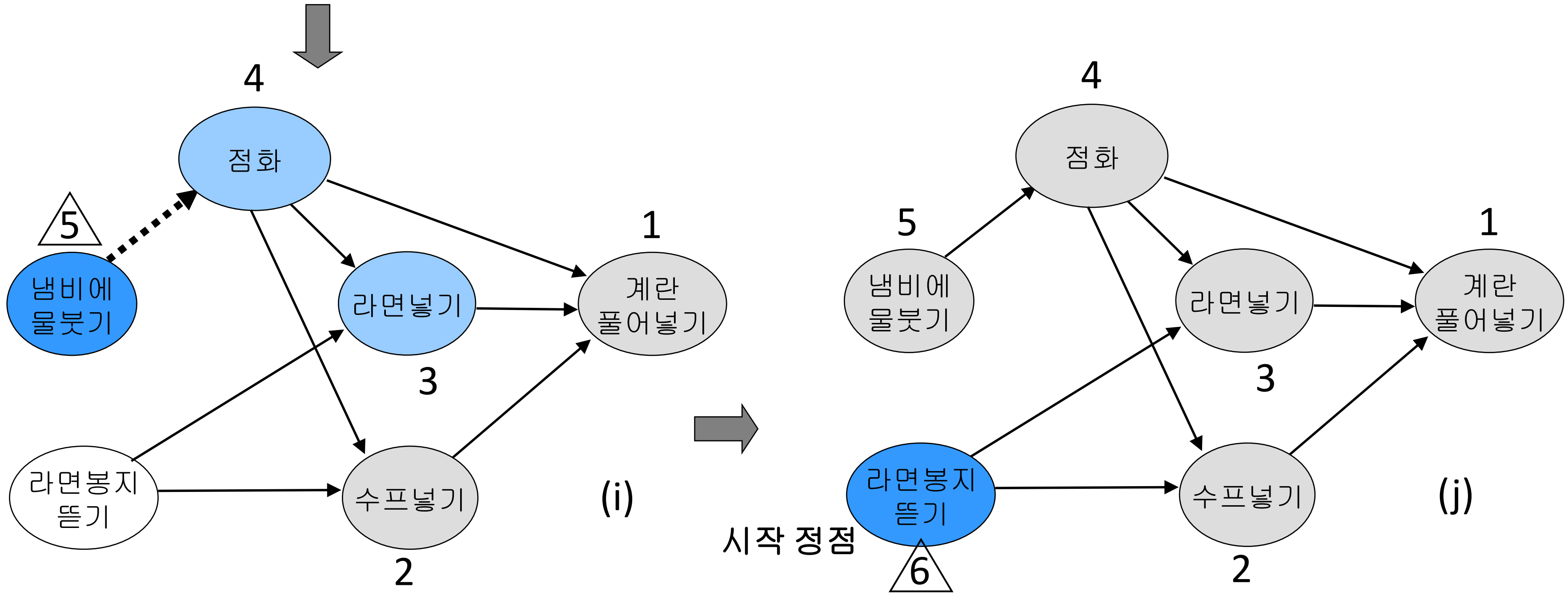
✓알고리즘 수행이 끝나고 나면 연결 리스트 R에는 정점들이 위상정렬된 순서로 매달려 있다.

✓수행시간: $\Theta(|V|+|E|)$

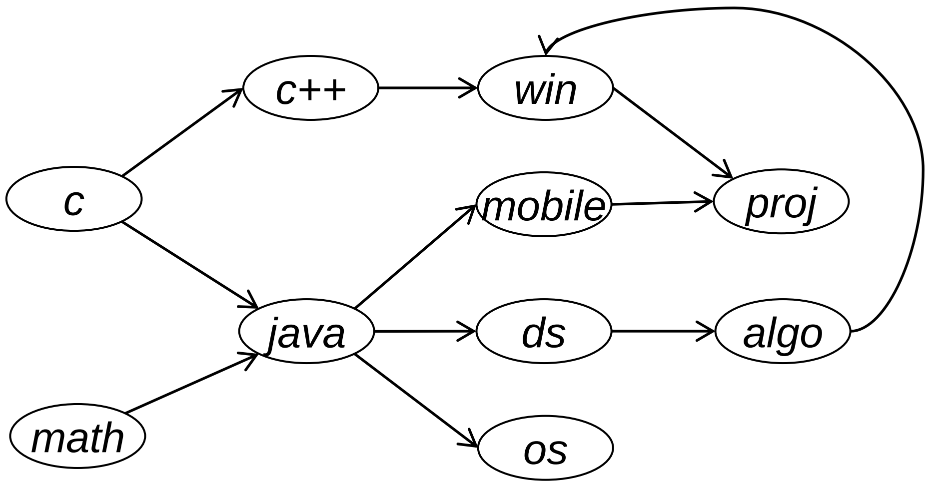
위상정렬 알고리즘 2의 작동 예




시작 정점
냄비에 물붓기
점화
라면넣기
계란 풀어넣기
라면봉지 뜯기
수프넣기
1
2
(e)
냄비에 물붓기
점화
라면넣기
계란 풀어넣기
라면봉지 뜯기
수프넣기
1
2
(f)
냄비에 물붓기
점화
라면넣기
계란 풀어넣기
라면봉지 뜯기
수프넣기
1
2
3
(g)
4
냄비에 물붓기
점화
라면넣기
계란 풀어넣기
라면봉지 뜯기
수프넣기
1
2
3
(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
시작 정점
6
수프넣기
2
(j)

문제: 다음은 과목의 선수 관계를 나타낸 그래프이다. 이 그래프를 topological sort 하여 수강 가능한 순서대로 과목명을 일렬로 나열하세요.



요약

- 위상 정렬(topological sort)을 이용하면 DAG로 표현한 작업의 선후 관계에 따라 작업들을 일렬로 나열할 수 있다.
- 위상 정렬 알고리즘에 깊이 우선 탐색(DFS)을 이용할 수 있다.

학습내용

1. 그래프
2. 그래프의 표현
3. 너비 우선 탐색(BFS)와 깊이 우선 탐색(DFS)
4. 최소 신장 트리
5. 위상 정렬
6. **최단 경로**
7. 강연결 요소

최단 경로(Shortest Paths)

- 조건
  - 간선 가중치가 있는 유향 그래프(directed weighted graph)
  - 그래프가 무향 그래프라면, 양쪽으로 유향 간선이 있는 유향 그래프로 생각할 수 있다.
    - 즉, 무향 간선 (i, j)를 유향 간선 $<i, j>$와 $<j, i>$로 간주
- 두 정점 사이의 최단 경로
  - 두 정점 사이의 경로 중 간선의 가중치 합이 최소인 경로
  - 간선의 가중치 합이 음인 cycle이 있으면 문제가 정의되지 않는다.

여러 가지 최단 경로 알고리즘

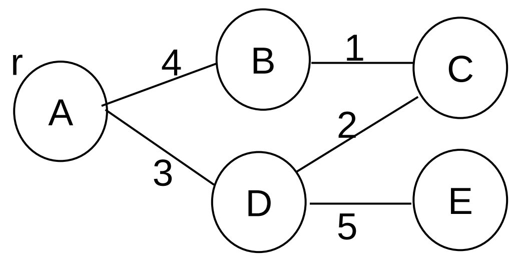
- 단일 시작점 최단 경로
  - 하나의 시작 정점으로부터 각 정점에 이르는 최단 경로를 구한다.
    - 다익스트라 알고리즘
      - 음의 가중치를 허용하지 않는 최단 경로
    - 벨만-포드 알고리즘
      - 음의 가중치를 허용하는 최단 경로
    - 사이클이 없는 그래프(DAG)의 최단 경로
- 모든 쌍 최단 경로
  - 모든 정점 쌍 사이의 최단 경로를 모두 구한다.
    - 플로이드-워샬 알고리즘

최단 경로 알고리즘

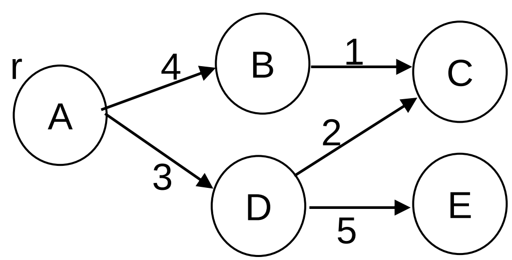
- 다익스트라 알고리즘
- 벨만-포드 알고리즘
- 플로이드-워샬 알고리즘
- DAG의 최단 경로

Dijkstra 알고리즘

- 단일 시작점 최단 경로 알고리즘
- 모든 간선 가중치는 음이 아니어야 함
- 최단 경로 구하는 방법
  - 최소 신장 트리를 위한 Prim 알고리즘과 원리가 비슷함
  - Prim 알고리즘에서는 d[v]가 정점 v를 신장 트리에 연결하는 최소 비용



  - Dijkstra 알고리즘에서는 d[v]가 시작 정점 r에서 정점 v에 이르는 최단 거리



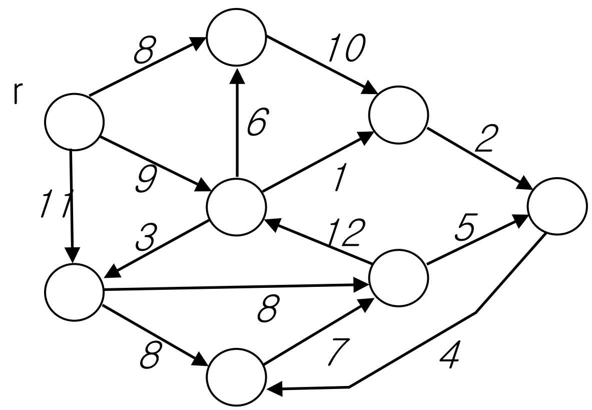
Dijkstra 알고리즘

```
Dijkstra(G, r)        ▷ G=(V, E): 주어진 그래프, r: 시작 정점
                      ▷ d[u] : 정점 r에서 정점 u에 이르는 최단 거리
{
    S ← Φ ;                       ▷ S : r로부터의 최단 거리가 확정된 정점 집합
    for each u∈V
            d[u] ← ∞ ;
    d[r] ← 0 ;
    while (S ≠ V) {              ▷ |V| 번 반복
        u ← extractMin(V-S, d) ; ▷ V-S : r로부터의 최단 거리가 미확정
        S ← S ∪ {u};
        for each v∈L(u)          ▷ L(u) : u의 인접 정점 집합
            if (v∈V-S and d[v] > d[u] + w(u, v)) then {
                d[v] ← d[u] + w(u, v) ;    이완(relaxation)
                prev[v] ← u ;
            }
    }
}

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

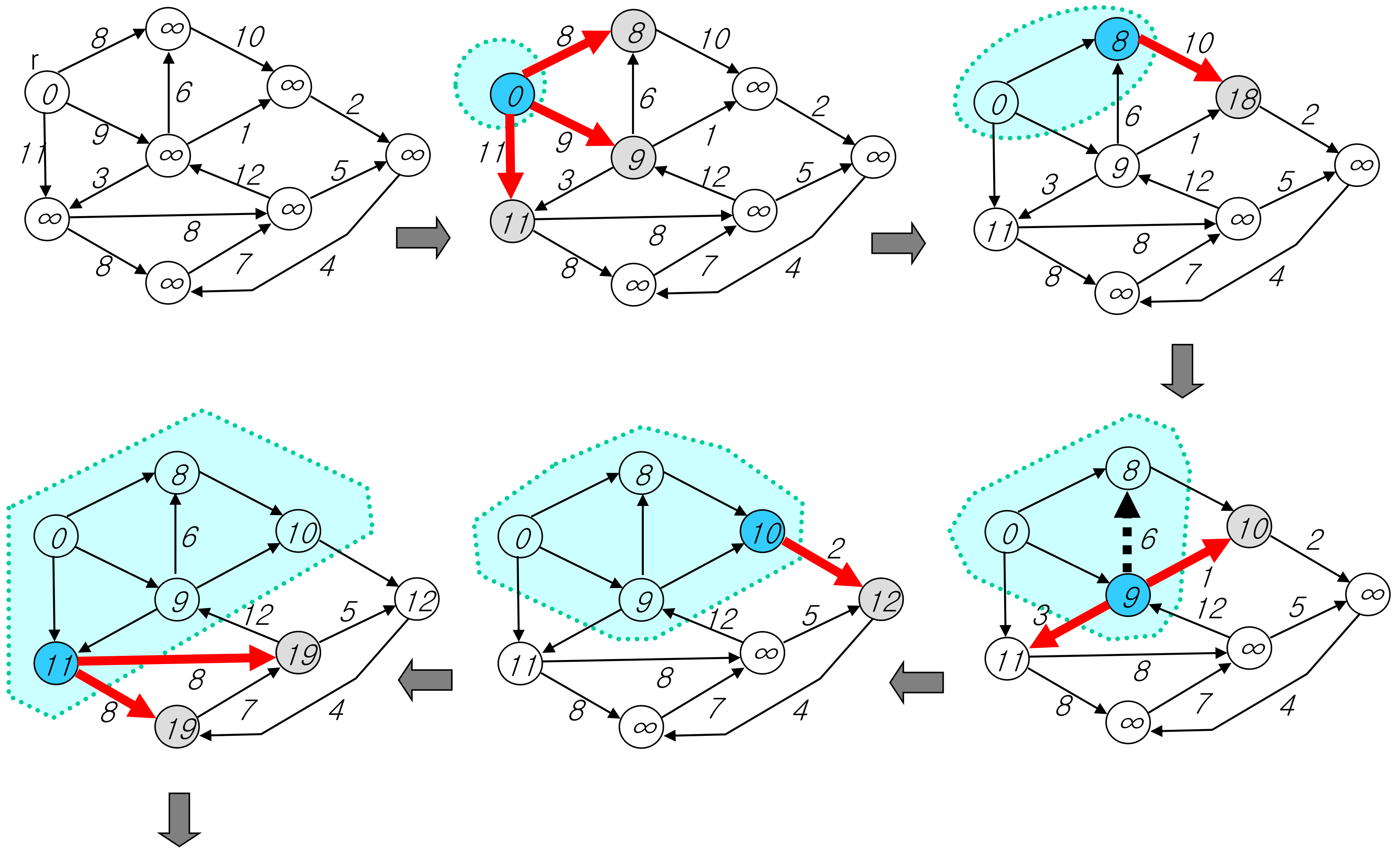
✓수행시간: $O(|E| log|V|)$

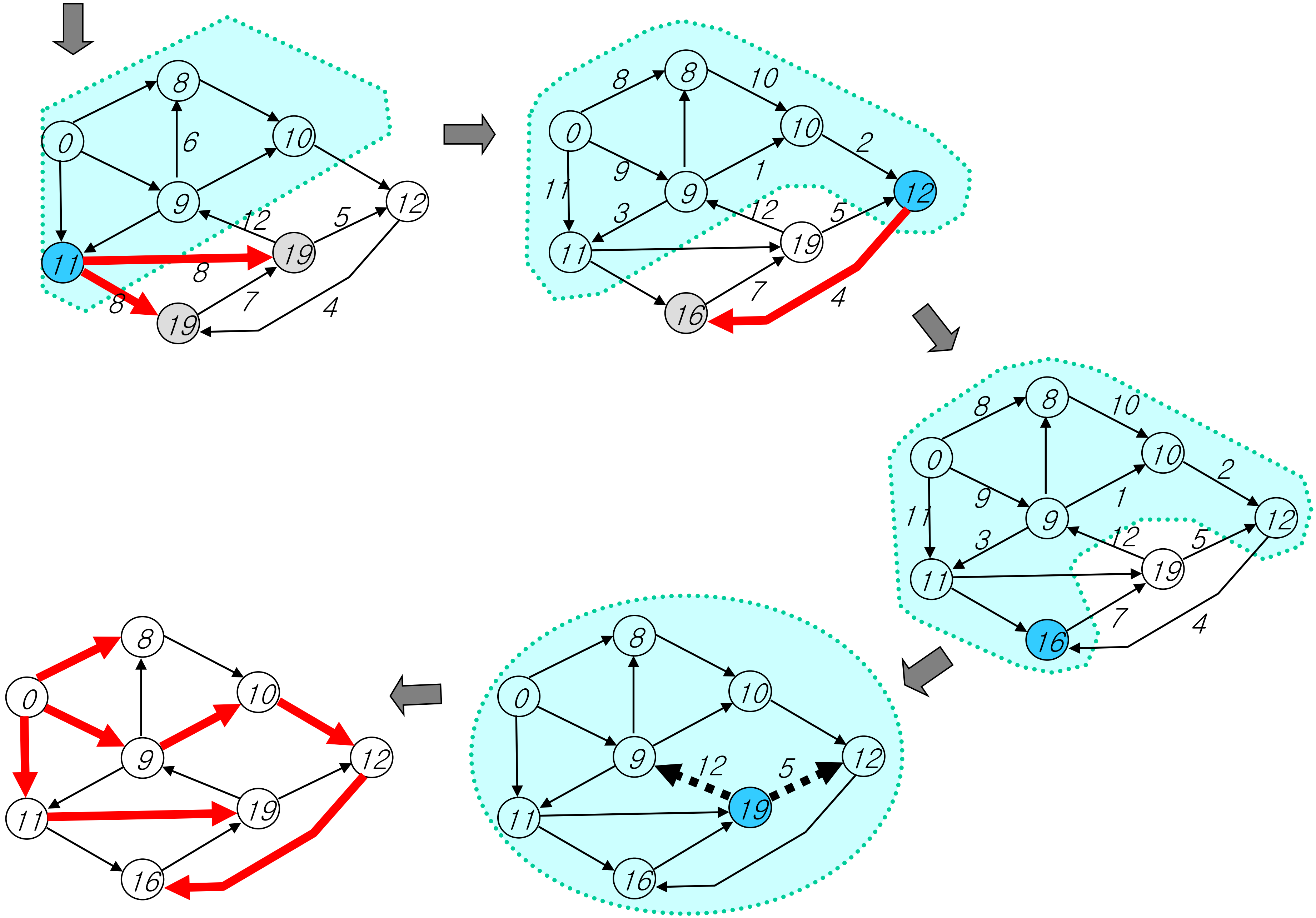
힙 이용

Dijkstra 알고리즘의 작동 예



Dijkstra 알고리즘의 작동 예





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

Bellman-Ford 알고리즘

- 단일 시작점 최단 경로 알고리즘
- 간선 가중치가 임의의 실수인 경우(음의 가중치 허용)
- 최단 경로 구하는 방법
  - 시작점으로부터 간선 최대 1개를 사용하는 최단 경로를 구함
  - 간선 최대 2개를 사용하는 최단 경로를 구함
  - 간선 최대 3개를 사용하는 최단 경로를 구함
  - …
  - 간선 최대 |V|-1개를 사용하는 최단 경로를 구함 → 음의 사이클이 존재하지 않으면 이것이 최종 해답임

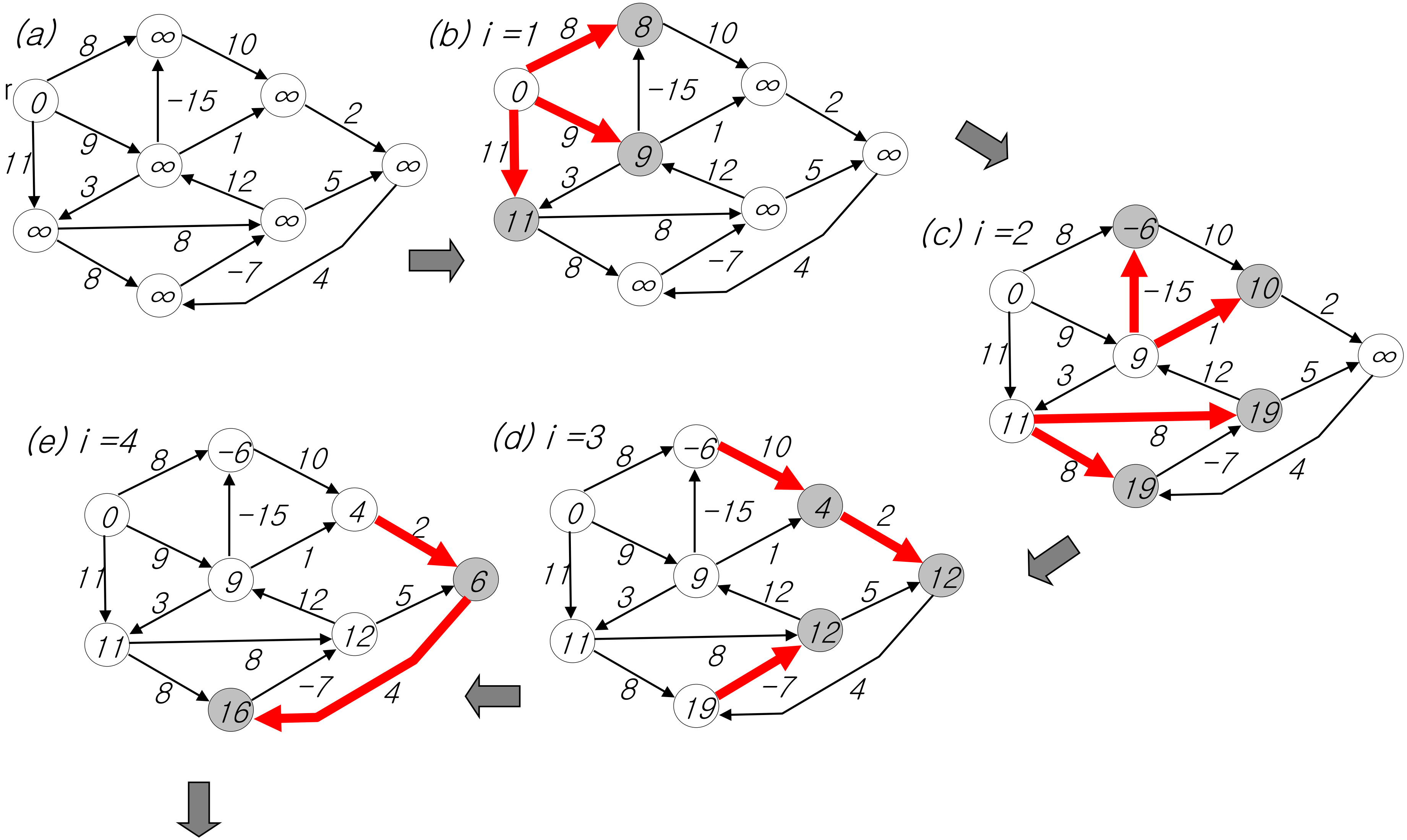
Bellman-Ford 알고리즘

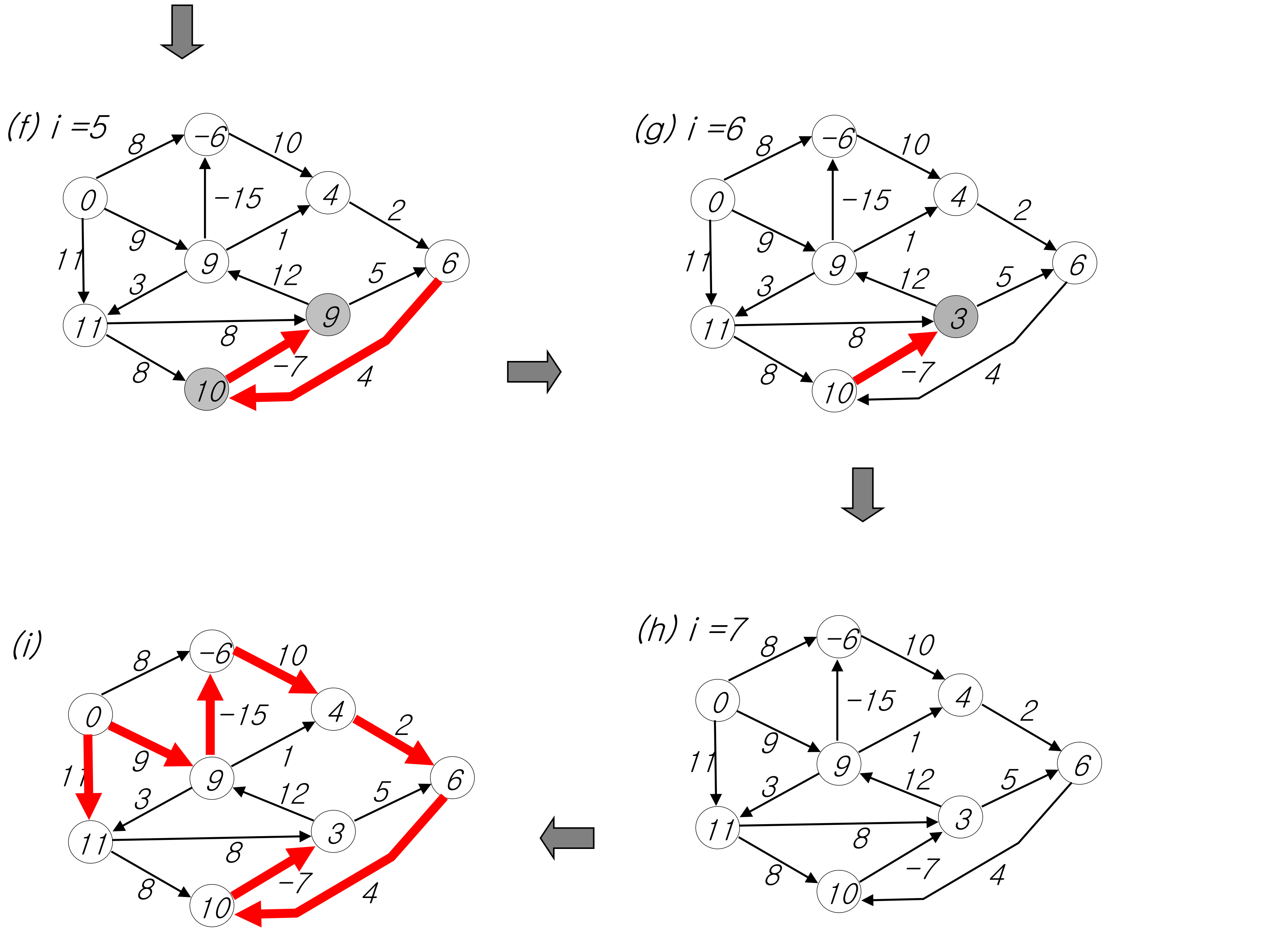
BellmanFord(G, r) ▷ G=(V, E): 주어진 그래프, r: 시작 정점
{
    for each $u \in V$
        d[u] ← ∞;
    d[r] ← 0;
    for i ← 1 **to** $|V|$-1
        for each $(u, v) \in E$
            if (d[v] > d[u] + $w(u, v)$) **then** {
                d[v] ← d[u] + $w(u, v)$; *이완(relaxation)*
                prev[v] ← u ;
            }

    ▷ 음의 사이클 존재 여부 확인
    for each $(u, v) \in E$
        if (d[v] > d[u] + $w(u, v)$) **then output** "해 없음"
}

✓수행시간: $O(|E||V|)$

Bellman-Ford 알고리즘의 작동 예



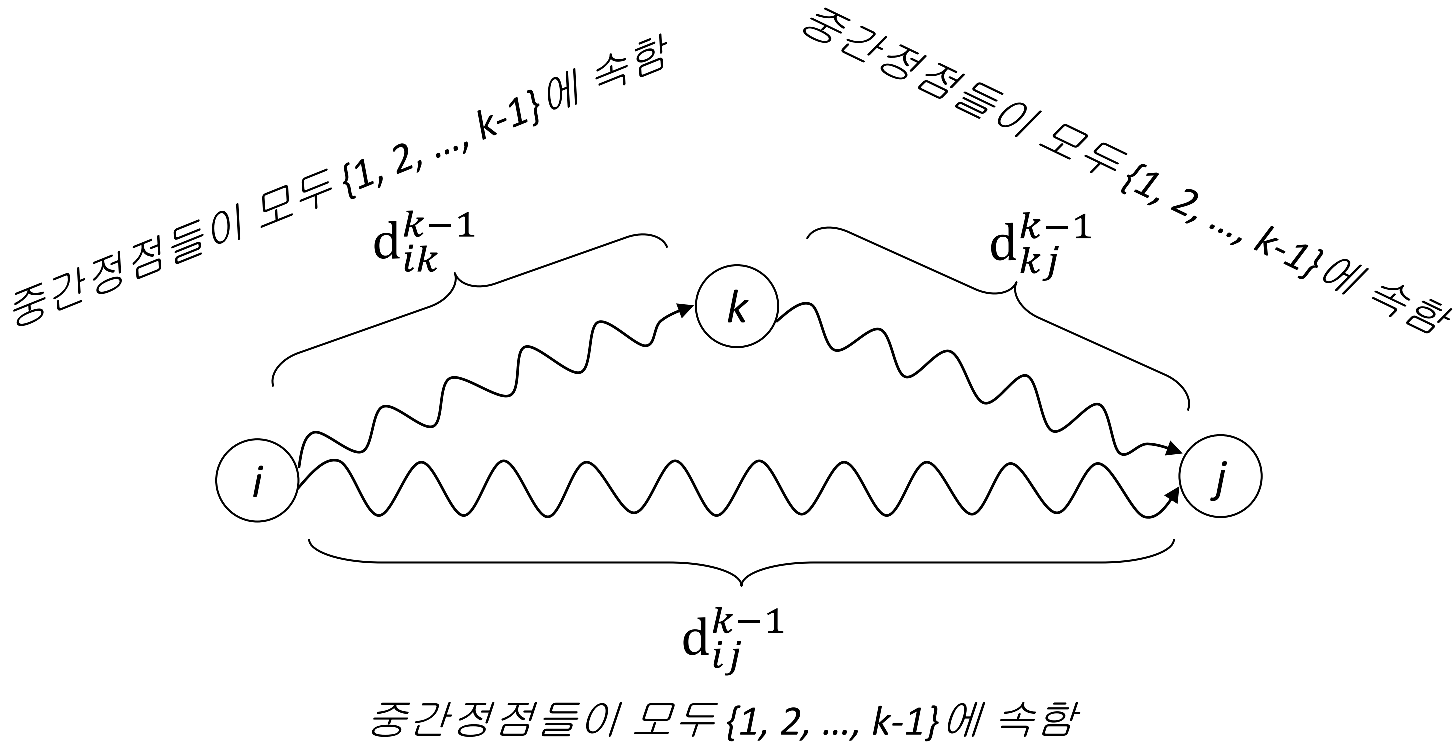

(f) i =5
(g) i =6
(h) i =7
(i)

Floyd-Warshall 알고리즘

- 모든 정점들간의 상호 최단경로 구하기
  - 구해야 할 최단 경로가 모두 n^2개 (n은 정점 수)
  - 동적 프로그래밍을 이용한 해결책인 플로이드-워샬 알고리즘은 $\Theta(n^3)$ 시간이 걸림

d^{k}_{ij} : 정점 집합 $\{v_1, v_2, \ldots, v_k\}$에 속하는 것들만 거쳐 v_i에서 v_j에 이르는 최단경로 길이

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



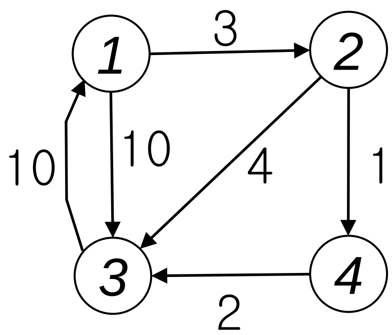
Floyd-Warshall 알고리즘

```
FloydWarshall(G) ▷ 그래프 G=(V, E)의 모든 정점들 간의 최단거리
{
    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};
}
```

d^{k}_{ij} : 정점 집합 $\{v_1, v_2, \ldots, v_k\}$에 속하는 것들만 거쳐
v_i에서 v_j에 이르는 최단경로 길이

✓수행시간: $\Theta(|V|^3)$

Floyd-Warshall 알고리즘 작동 예



d^{k}_{ij}

	1	2	3	4
1				
2				
3				
4				

d^{k}_{ij}

	1	2	3	4
1				
2				
3				
4				

DAG의 Shortest Path

- 사이클이 없는 유향 그래프를 DAG라고 한다.
  - DAG: Directed Acyclic Graph
- DAG의 단일 시작점 최단 경로
  - DAG에서의 최단경로는 선형시간에 간단히 구할 수 있다.

DAG의 최단경로 알고리즘

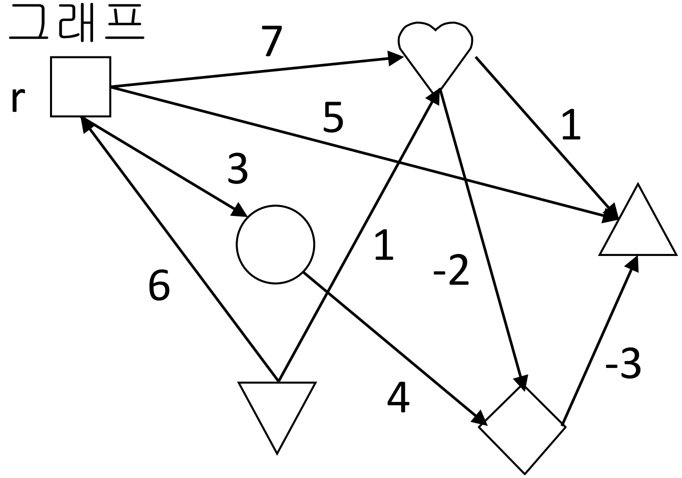
DAG-ShortestPath(G, r) ▷ G=(V, E): 주어진 그래프, r: 시작 정점
▷ d[u] : r로부터 u까지의 최단경로 길이

{
  for each $u \in V$
    d[u] $\leftarrow \infty$;
  d[r] $\leftarrow$ 0;
  G의 정점들을 위상정렬한다; ▷ $\Theta(|V|+|E|)$
  for each $u \in V$ (위상정렬 순서로)
    for each $v \in L(u)$ ▷ $L(u)$: u의 인접 정점 집합
      if (d[v] > d[u] + $w(u, v)$) **then** { ▷중첩루프 통틀어 |E|번 수행
        d[v] $\leftarrow$ d[u] + $w(u, v)$;
        prev[v] $\leftarrow$ u ;
      }
  }
}

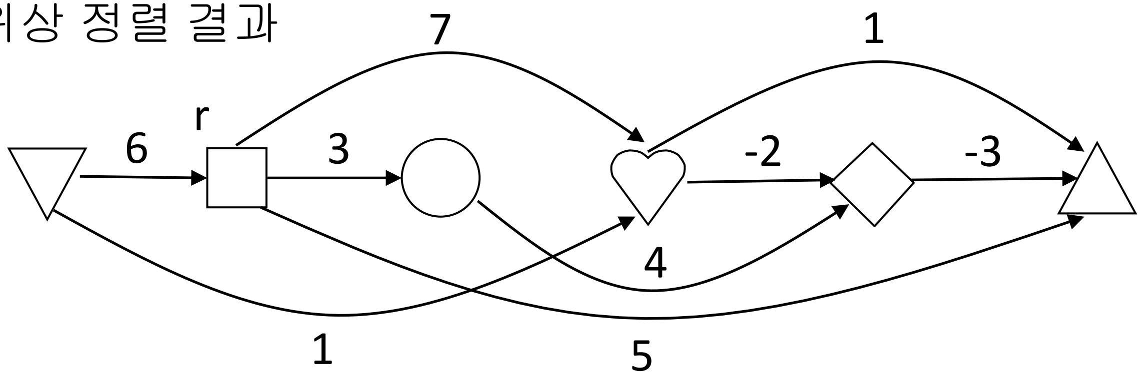
✓수행시간: $\Theta(|V|+|E|)$

DAG-ShortestPath 작동 예

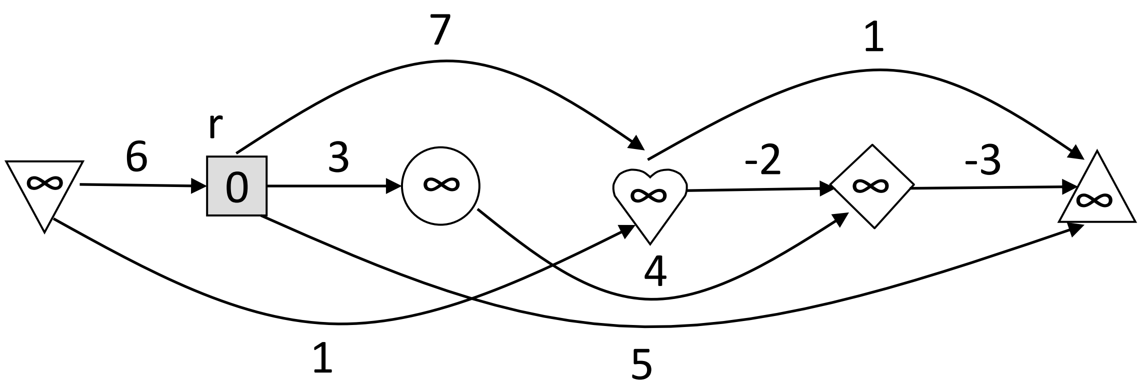
(a) 주어진 그래프



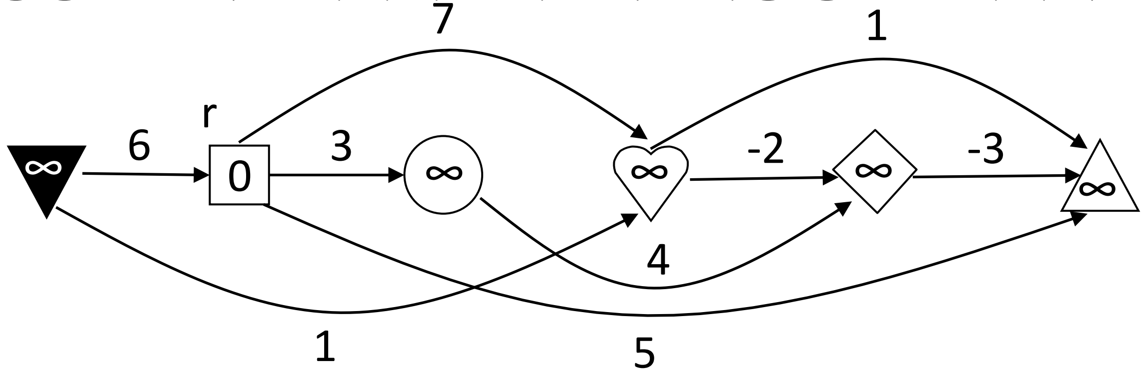
(b) 위상 정렬 결과



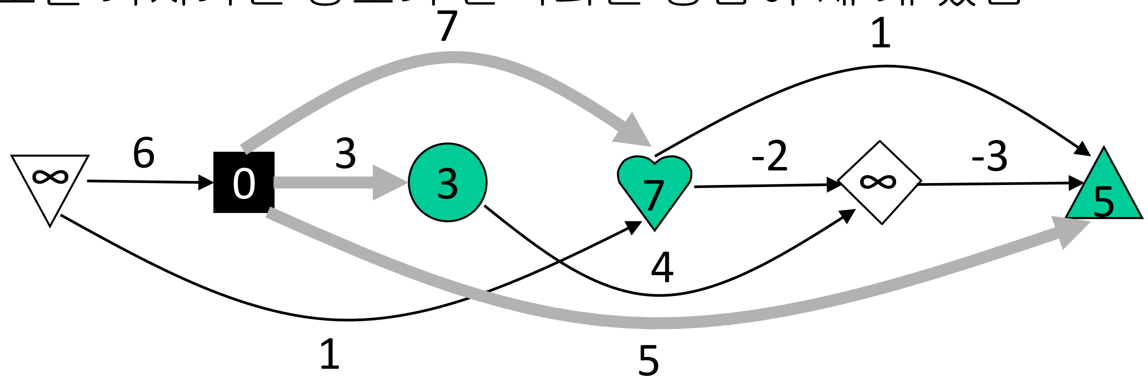
(c) r로부터의 최단경로 길이를 초기화



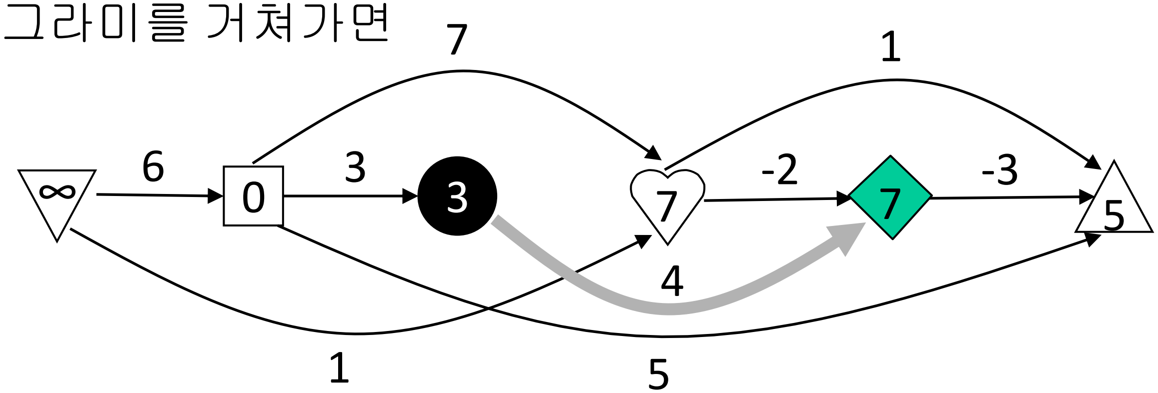
(d) 위상정렬 순서로 하나씩 검사 – 역삼각형 정점을 거쳐서 가도 경로 단축되지 않음



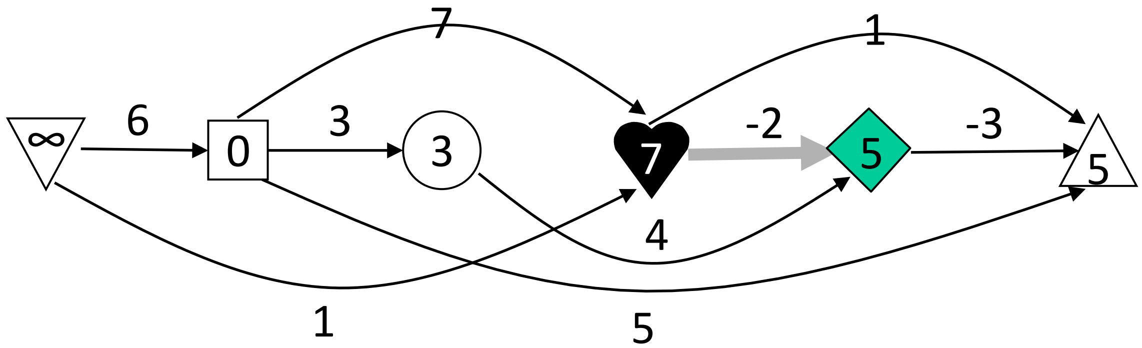
(e) 네모를 거쳐가면 경로가 단축되는 정점이 세 개 있음



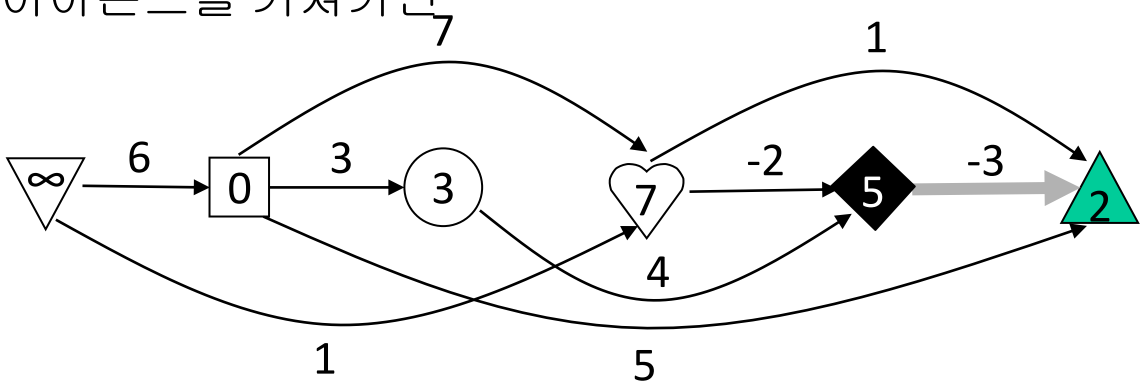
(f) 동그라미를 거쳐가면



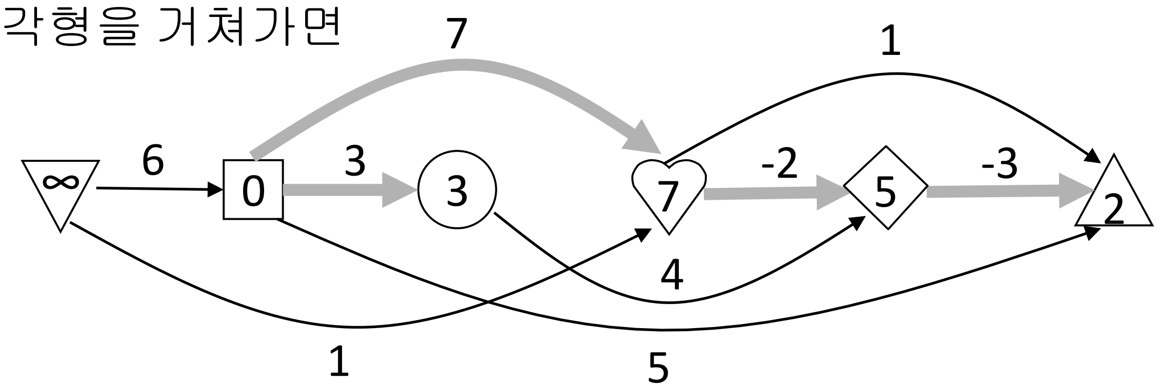
(g) 하트를 거쳐가면



(h) 다이아몬드를 거쳐가면



(i) 삼각형을 거쳐가면



요약

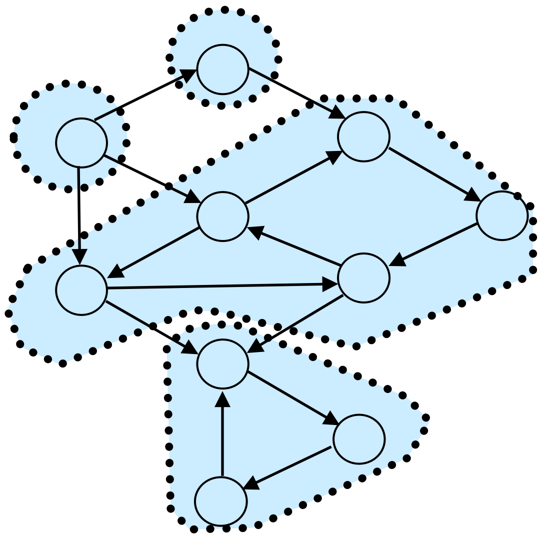
- 최단 경로 알고리즘에는 하나의 시작점부터 모든 정점에 이르는 최단 경로를 구하는 유형과 모든 정점쌍 간의 최단 경로를 구하는 유형이 있다.
- 다익스트라 알고리즘
- 벨만-포드 알고리즘
- 플로이드-워샬 알고리즘
- DAG의 최단 경로는 선형 시간에 구할 수 있음

학습내용

1. 그래프
2. 그래프의 표현
3. 너비 우선 탐색(BFS)와 깊이 우선 탐색(DFS)
4. 최소 신장 트리
5. 위상 정렬
6. 최단 경로
7. **강연결 요소**

강연결 요소

- "강하게 연결됨(strongly connected)"이란?
  - 유향 그래프 G의 모든 정점쌍에 대해서 양방향으로 경로가 존재하면 G는 강하게 연결되었다고 한다.
- 강하게 연결된 부분 그래프(subgraph)를 **강연결 요소(Strongly Connected Component)**라고 한다.
  - 예) 오른쪽 그래프는 4개의 강연결요소를 갖는다.



- 그래프의 강연결요소들을 찾는 알고리즘을 알아보자.
  - 깊이우선탐색(DFS)을 이용

강연결 요소 구하기 알고리즘

stronglyConnectedComponent(G)

{

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

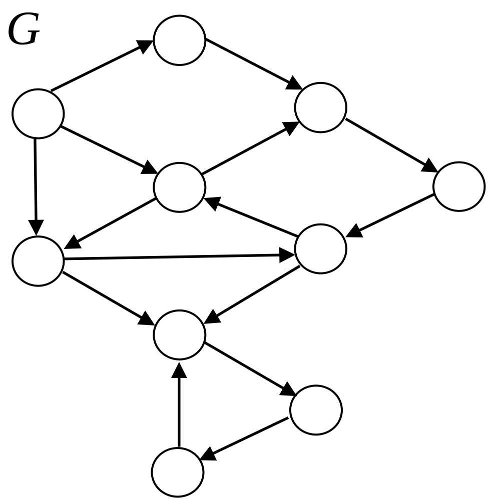
2) G의 모든 간선들의 방향을 뒤집어 G^R을 만든다;

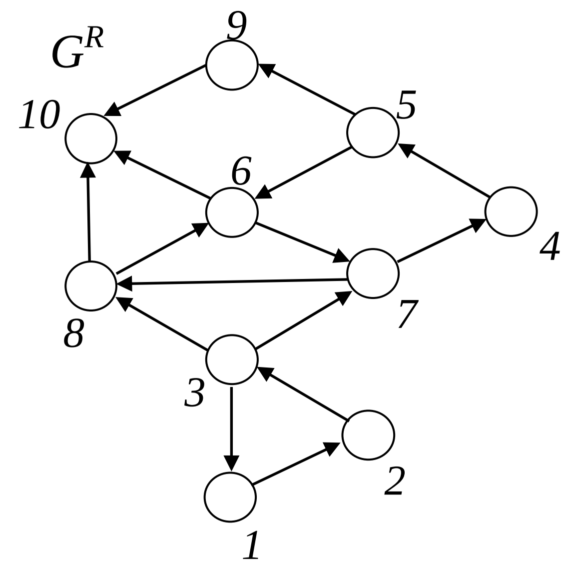
3) G^R에 대해 DFS를 수행하되 시작점은 단계 1에서 구한 $f[v]$가 가장 큰 정점으로 잡는다;

4) 단계 3에서 만들어진 분리된 트리 각각을 강연결요소로 리턴한다;

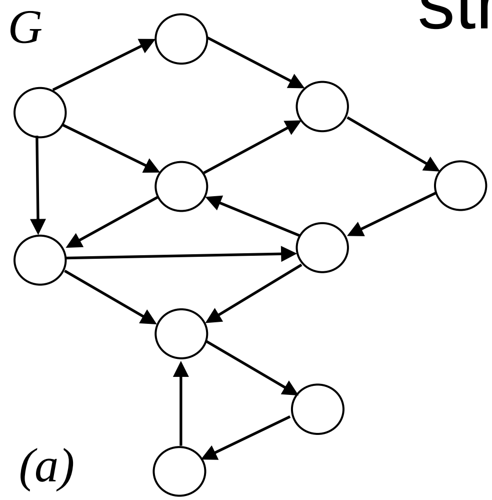
}

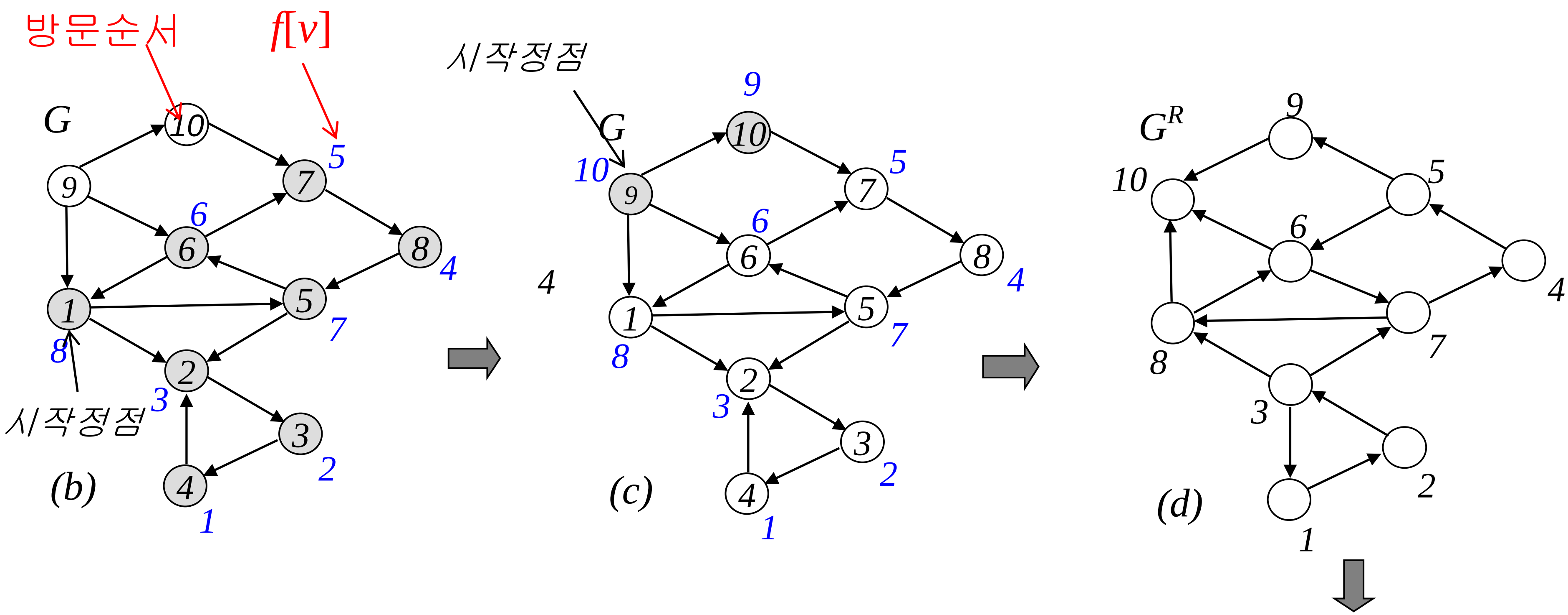
✓수행시간: $\Theta(|V|+|E|)$

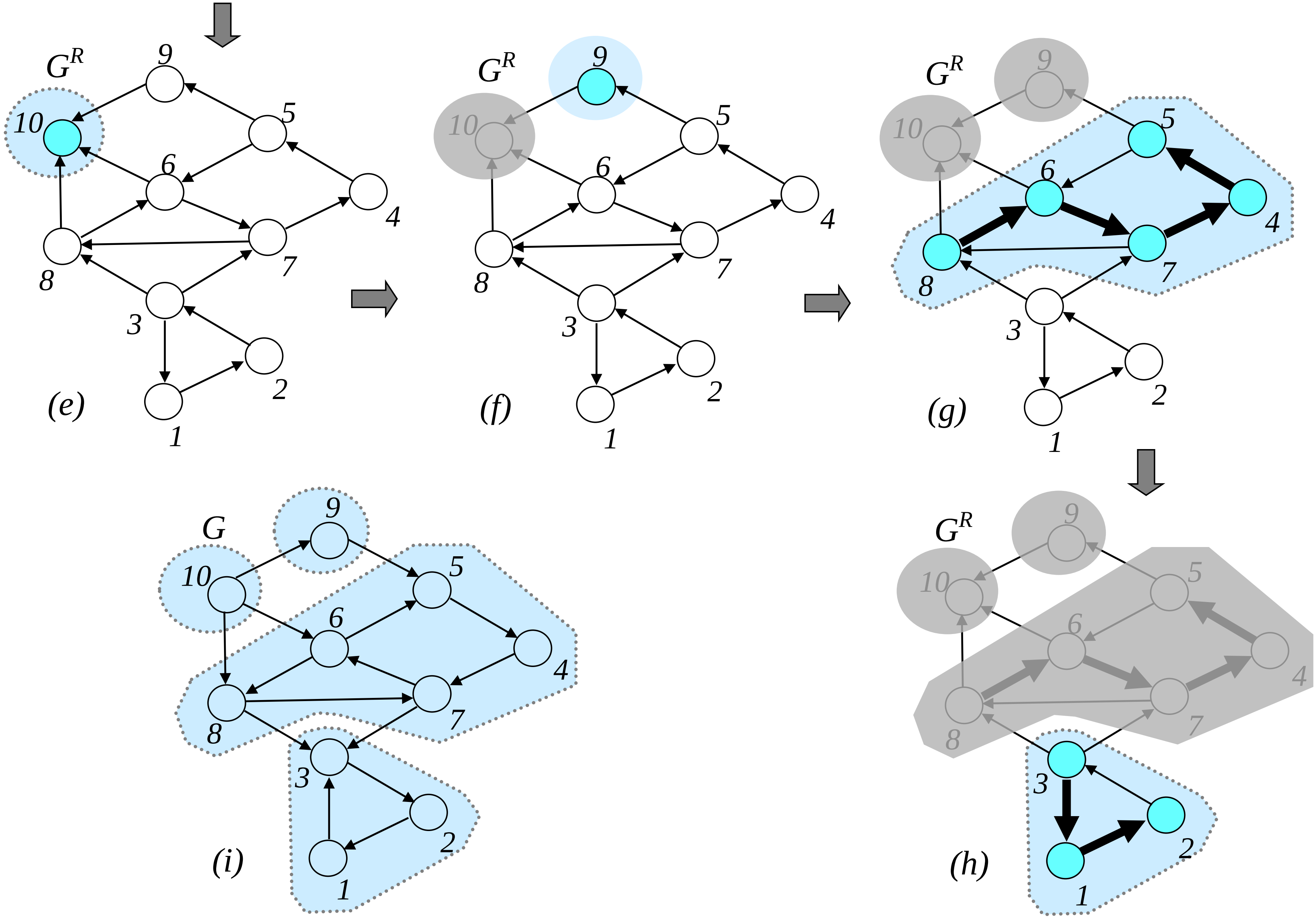

G


G^R
9
10
5
6
4
8
7
3
2
1

stronglyConnectedComponent 작동 예







최종 결과: G가 4개의 강연결요소로 분리됨

요약

- 유향 그래프에서 모든 정점 쌍에 대해 양방향 경로가 존재하면 그래프가 강하게 연결되었다고 한다.
- 그래프에서 강하게 연결된 부분 그래프를 강연결 요소(Strongly Connected Component)라고 한다.
- 임의의 그래프에서 강연결 요소들을 찾는 문제는 깊이 우선 탐색(DFS)을 이용하는 대표적인 응용 중 하나이다.