그래프Graph

그래프란
그래프의 표현
그래프 순회
최소 신장 트리
위상 정렬
최단 경로

1. 그래프란

그래프Graph

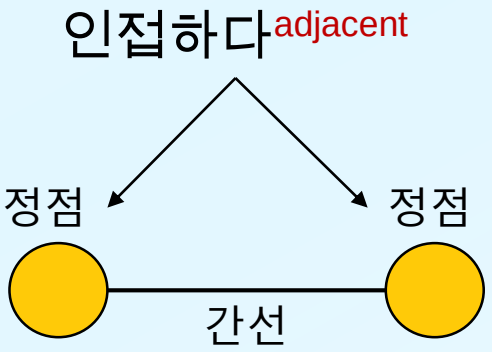
그래프 $G = (V, E)$

V : 정점vertex, node들의 집합

E : 간선edge들의 집합

정점: 대상(대상물, 개념 등)

간선: 대상들간의 관계를 나타냄



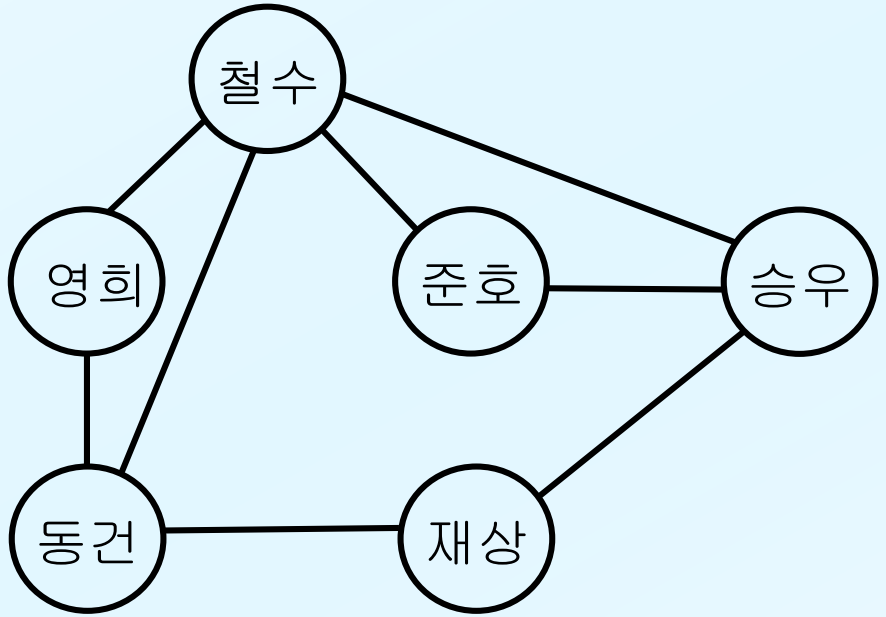
간선으로 연결된 두 정점은 관계가 있다.

이들을 인접하다Adjacent고 한다

그래프의 종류

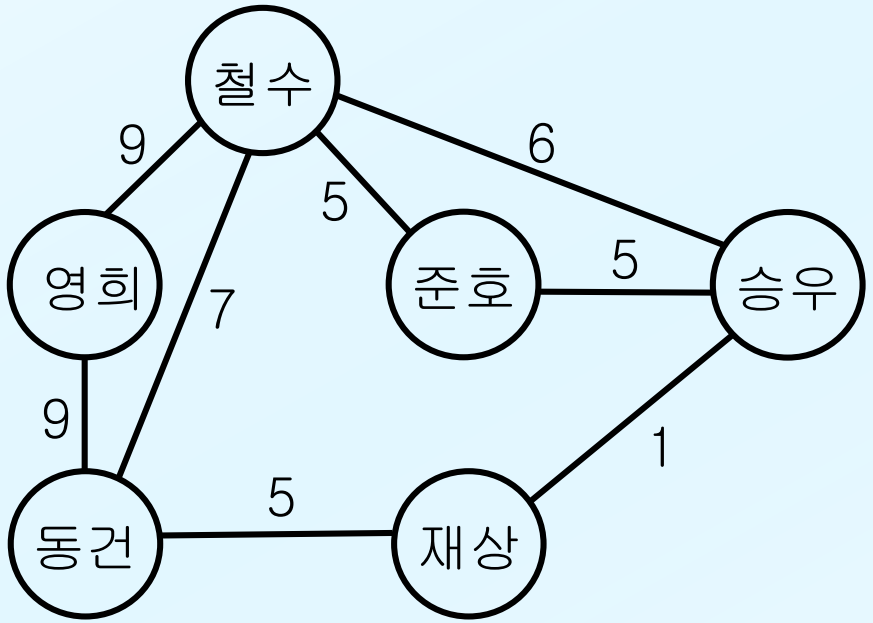
기본적인 그래프

무향 그래프라고도 한다



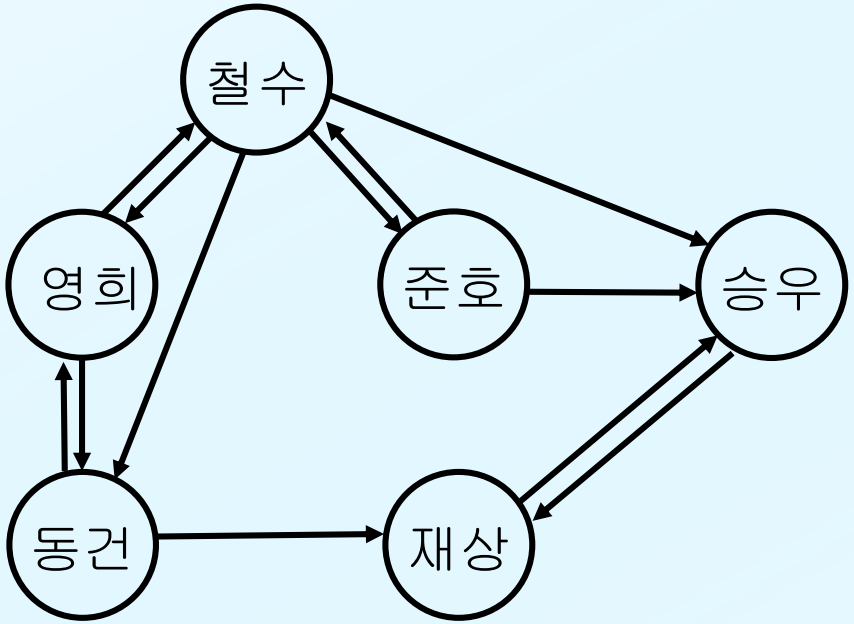
사람들간의 친분 관계를 나타낸 그래프

가중 그래프Weighted Graph



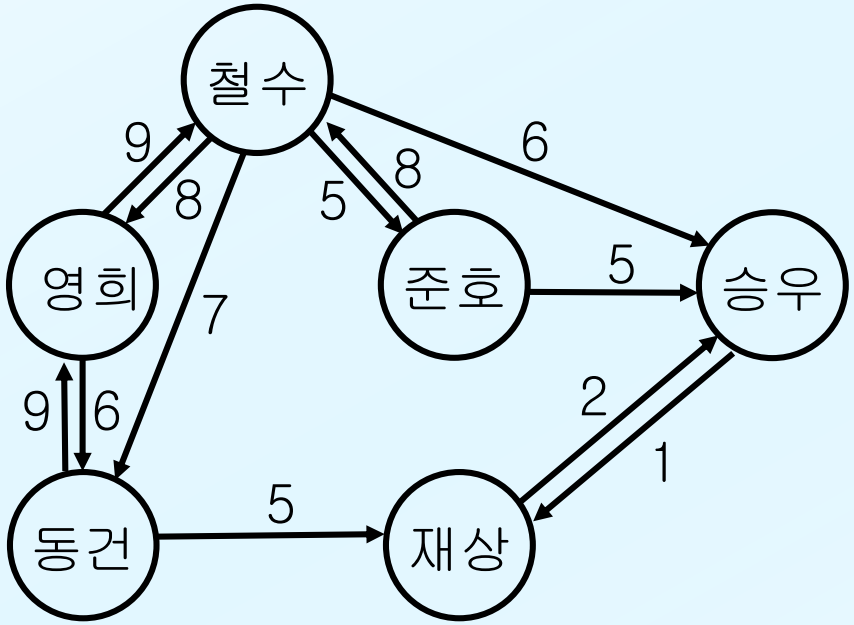
친밀도를 가중치로 나타낸 친분관계 그래프

방향 그래프(유향 그래프)Directed Graph



방향을 고려한 친분관계 그래프

가중 방향 그래프Weighted Directed Graph



방향을 고려하고 친밀도를 가중치로 나타낸 그래프

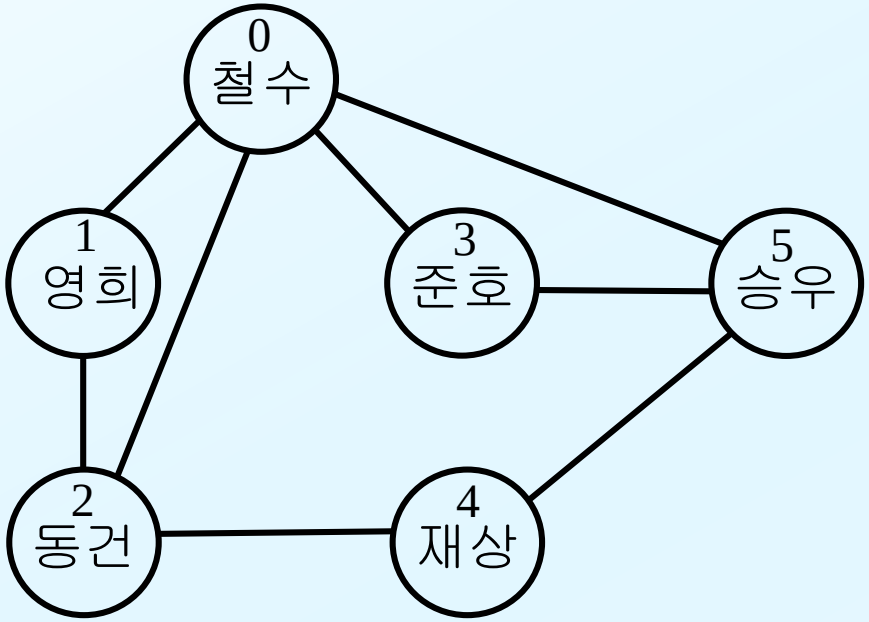
2. 그래프의 표현

인접 행렬Adjacency Matrix

그래프가 n*n 행렬 A[][]로 표현된다.

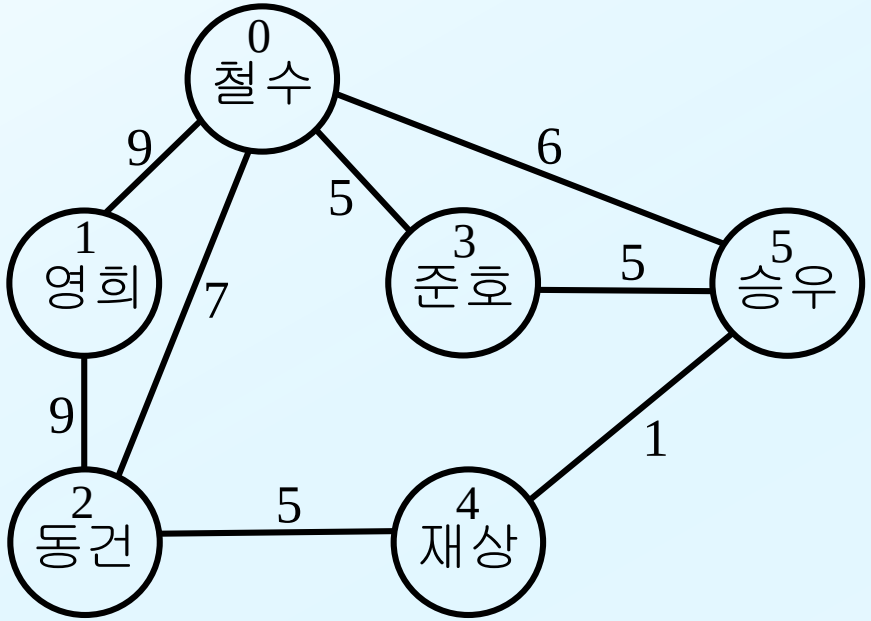
정점 i와 정점 j 사이에 간선의 존재 여부를 A[*i*][*j*]에 기록한다

무향 그래프의 인접 행렬



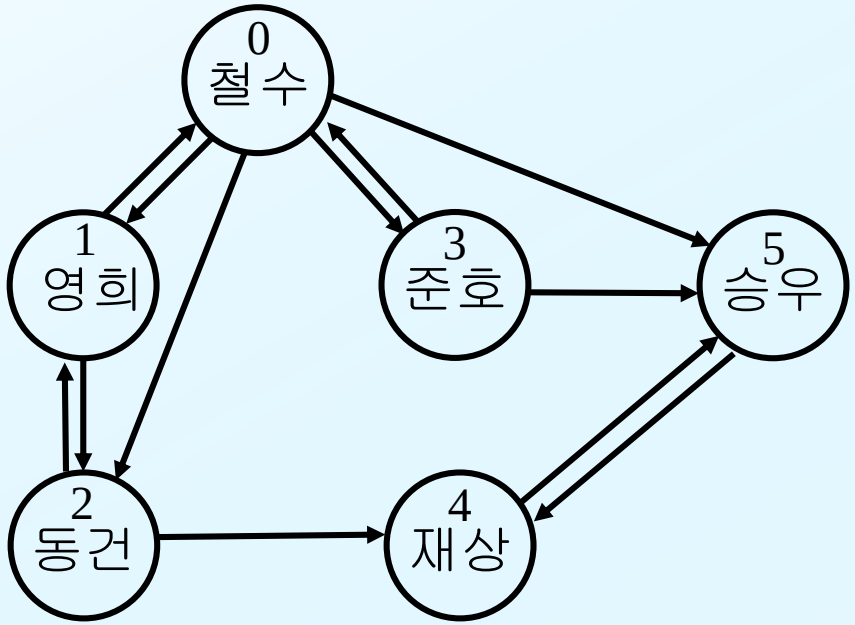
	0	1	2	3	4	5
0	0	1	1	1	0	1
1	1	0	1	0	0	0
2	1	1	0	0	1	0
3	1	0	0	0	0	1
4	0	0	1	0	0	1
5	1	0	0	1	1	0

가중 무향 그래프의 인접 행렬



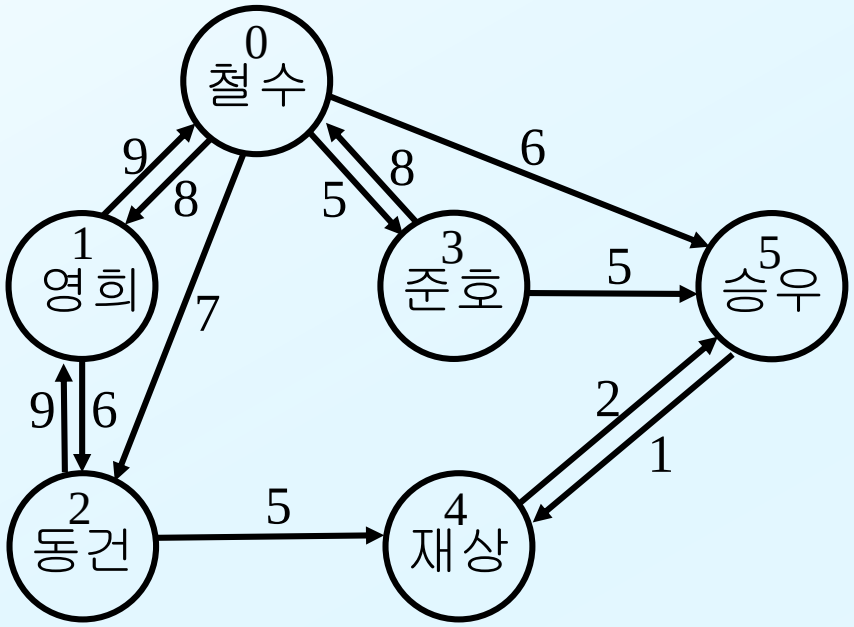
	0	1	2	3	4	5
0	0	9	7	5	0	6
1	9	0	9	0	0	0
2	7	9	0	0	5	0
3	5	0	0	0	0	5
4	0	0	5	0	0	1
5	6	0	0	5	1	0

방향 그래프의 인접 행렬



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

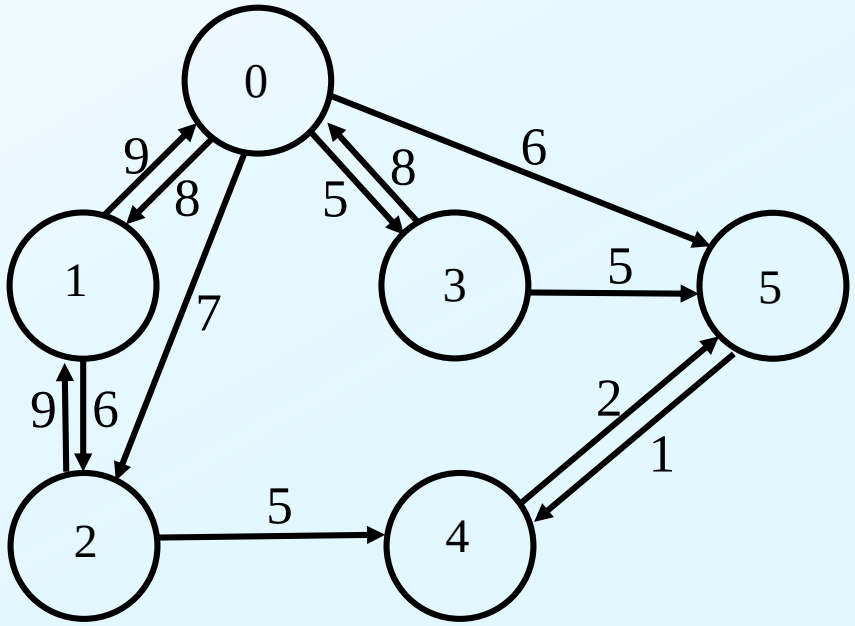
가중 방향 그래프의 인접 행렬



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

때로는..

인접하지 않은 정점 간의 가중치는 0이 아닌 ∞가 더 적합할 수도 있다



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

이것이 도시들간의 도로 상태(길이)를 나타낸다면

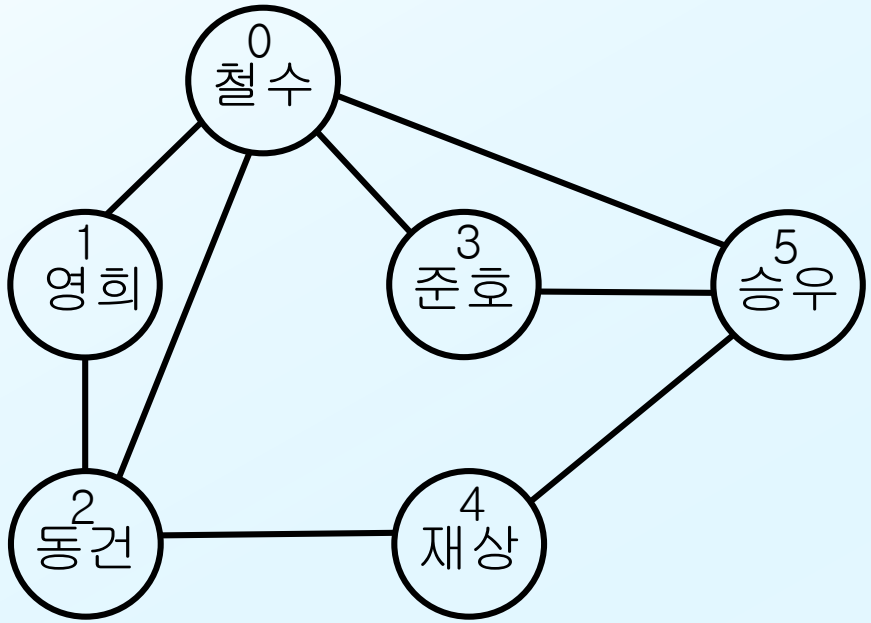
인접 리스트Adjacency List

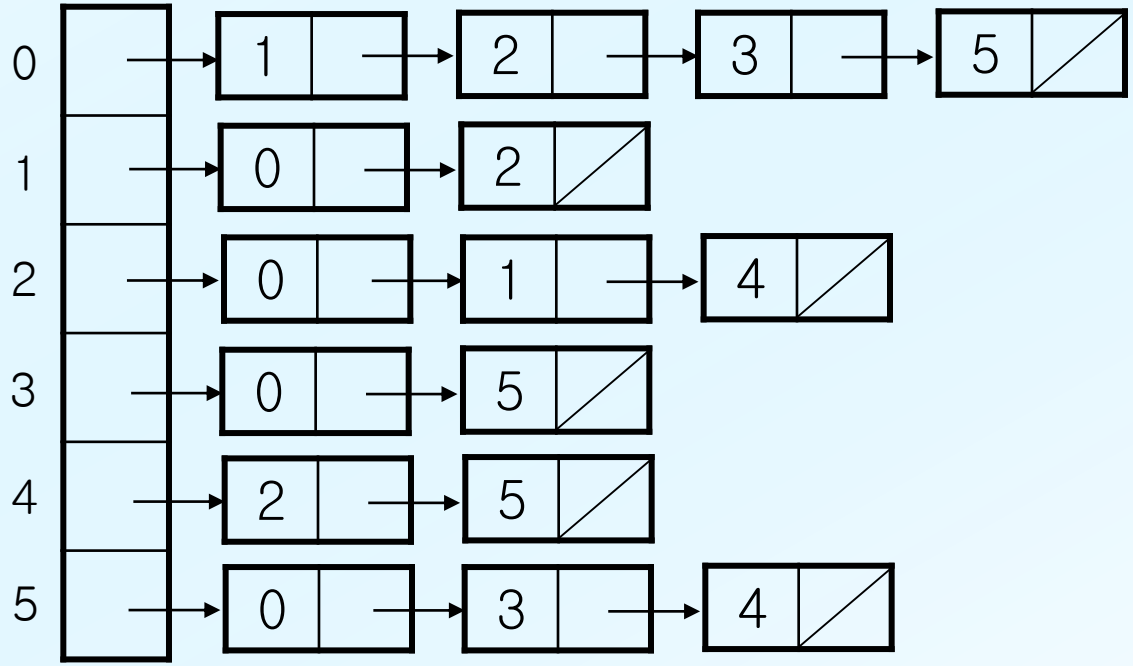
그래프가 n 개의 연결 리스트로 표현된다.

정점 i에 연결된 정점들은 연결 리스트 i로 관리된다

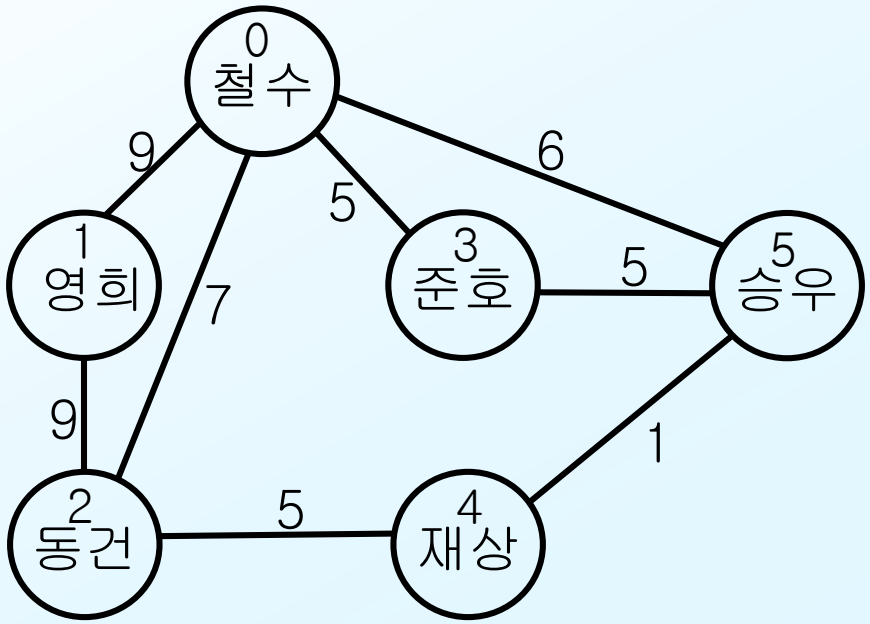
가중치가 있으면 함께 기록한다

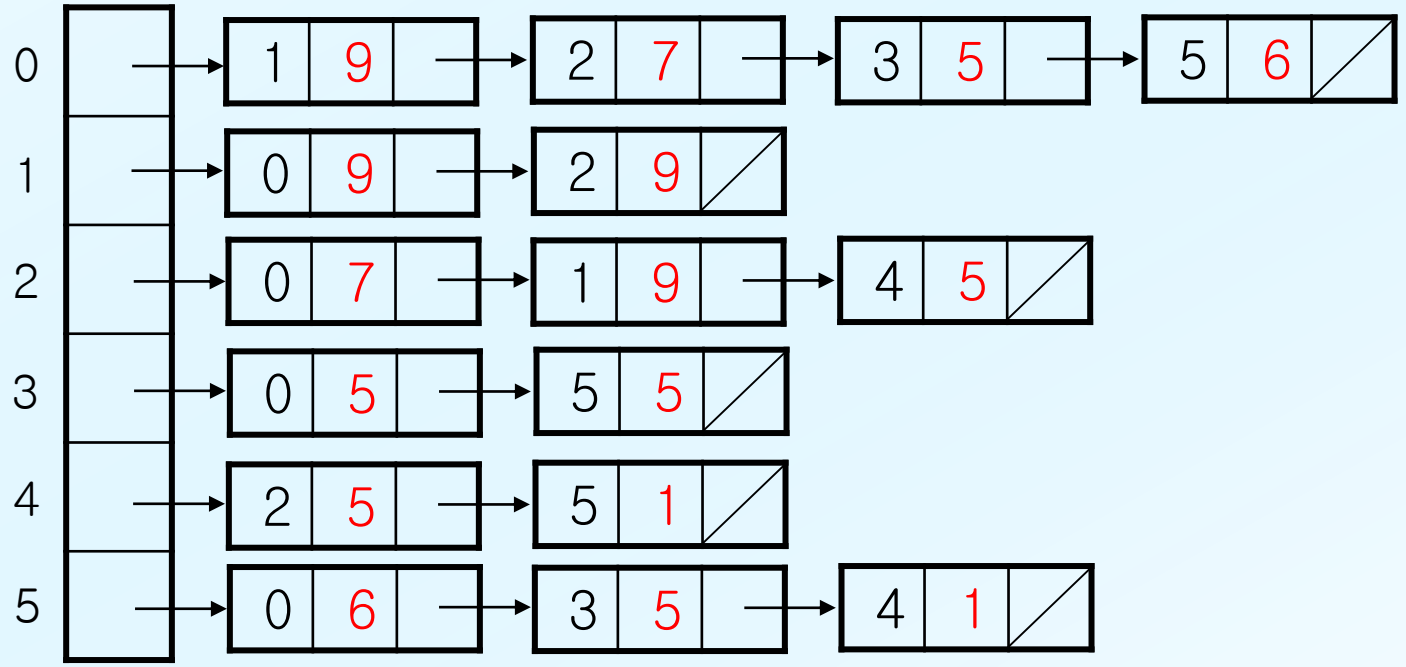
무향 그래프의 인접 리스트



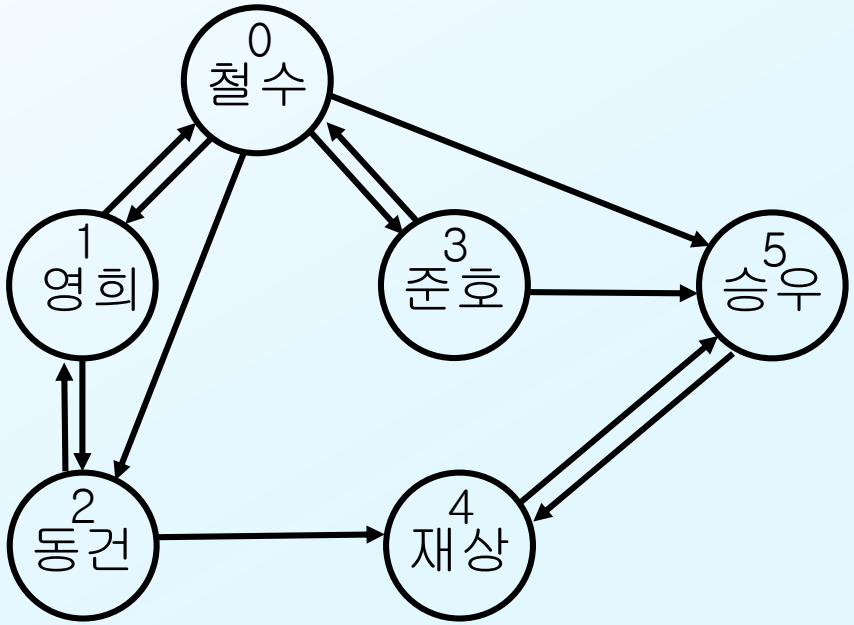


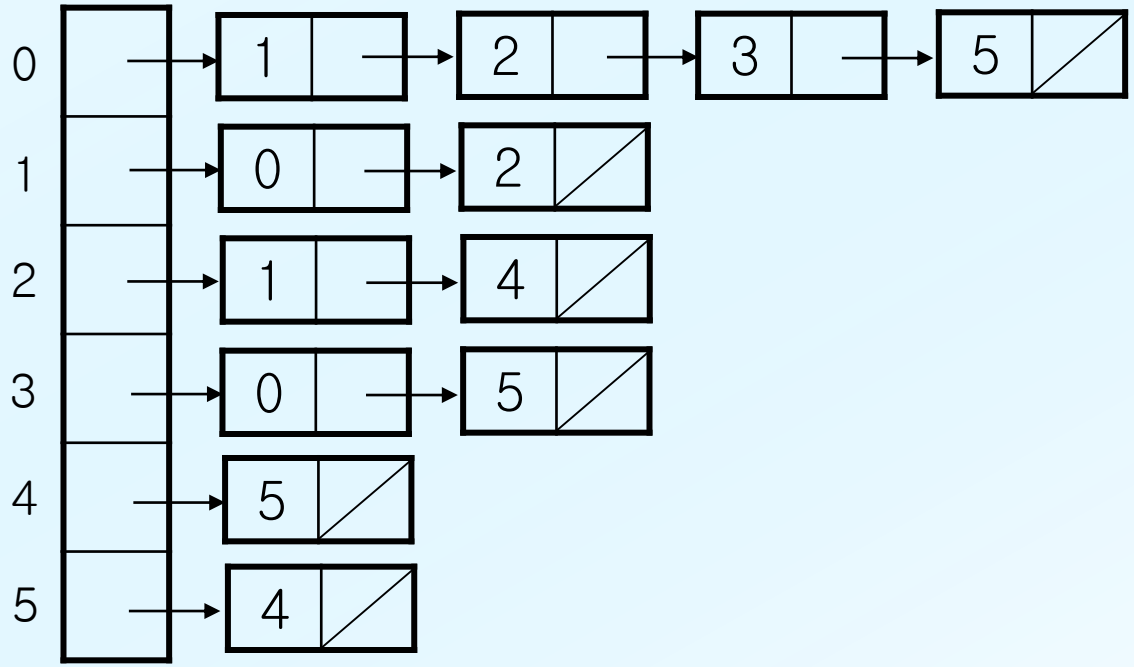
가중 그래프의 인접 리스트





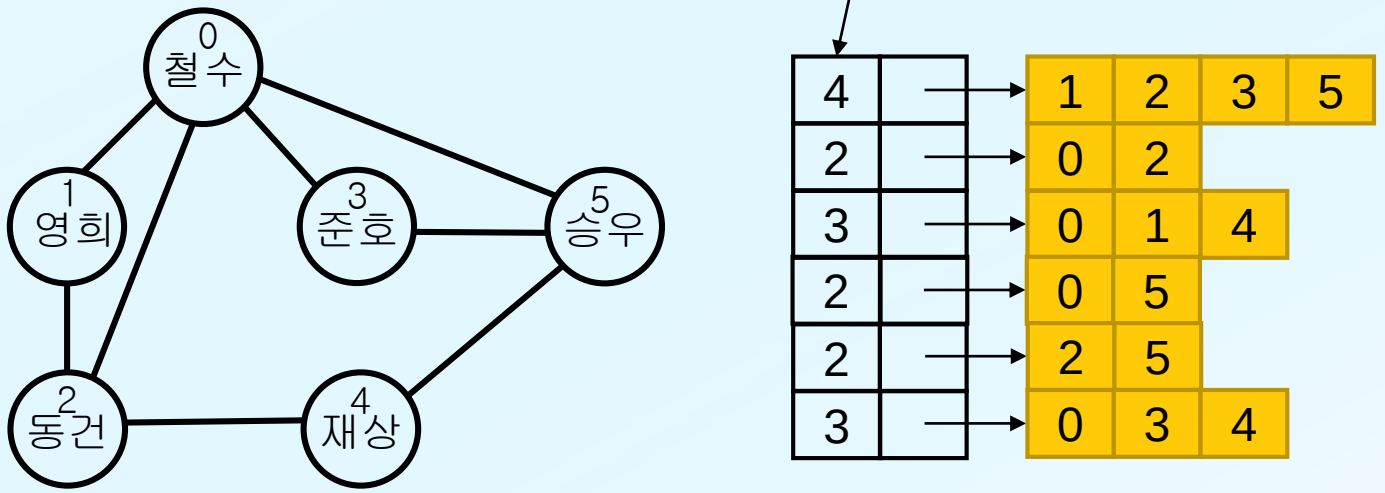
방향 그래프의 인접 리스트





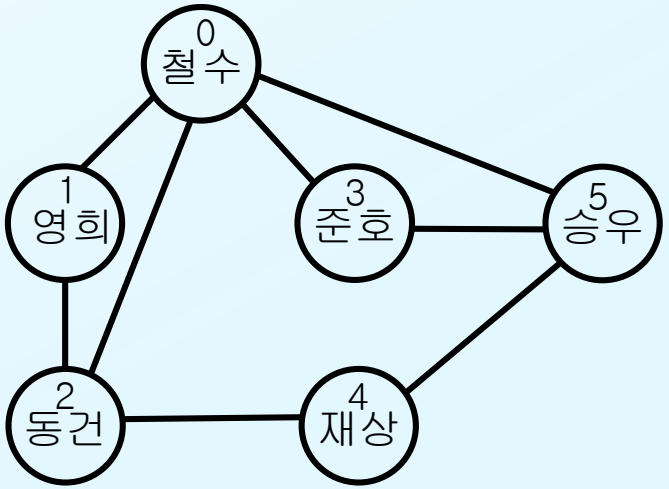
인접 배열Adjacency Array

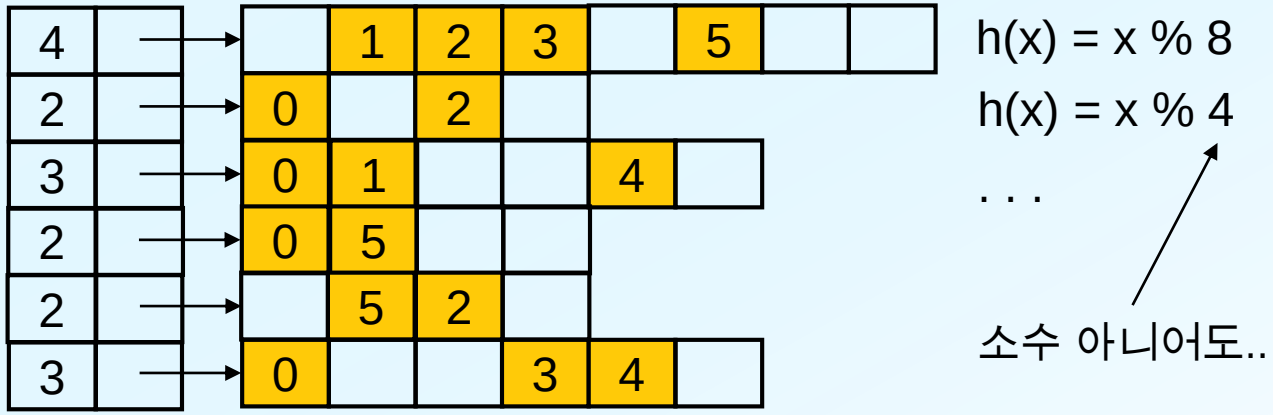
그래프가 n 개의 배열(리스트)로 표현된다
필요하면 가중치도 함께 기록한다



인접 해시 테이블Adjacency Hash Table

그래프가 n 개의 해시 테이블로 표현된다
필요하면 가중치도 함께 기록한다





3. 그래프 순회

그래프 순회 Graph Traversal

한 정점에서 시작하여 도달 가능한 모든 정점을 방문한다

대표적인 두 알고리즘

- 너비 우선 탐색 Breadth-First Search
- 깊이 우선 탐색 Depth-First Search

너비 우선 탐색Breadth-First Search

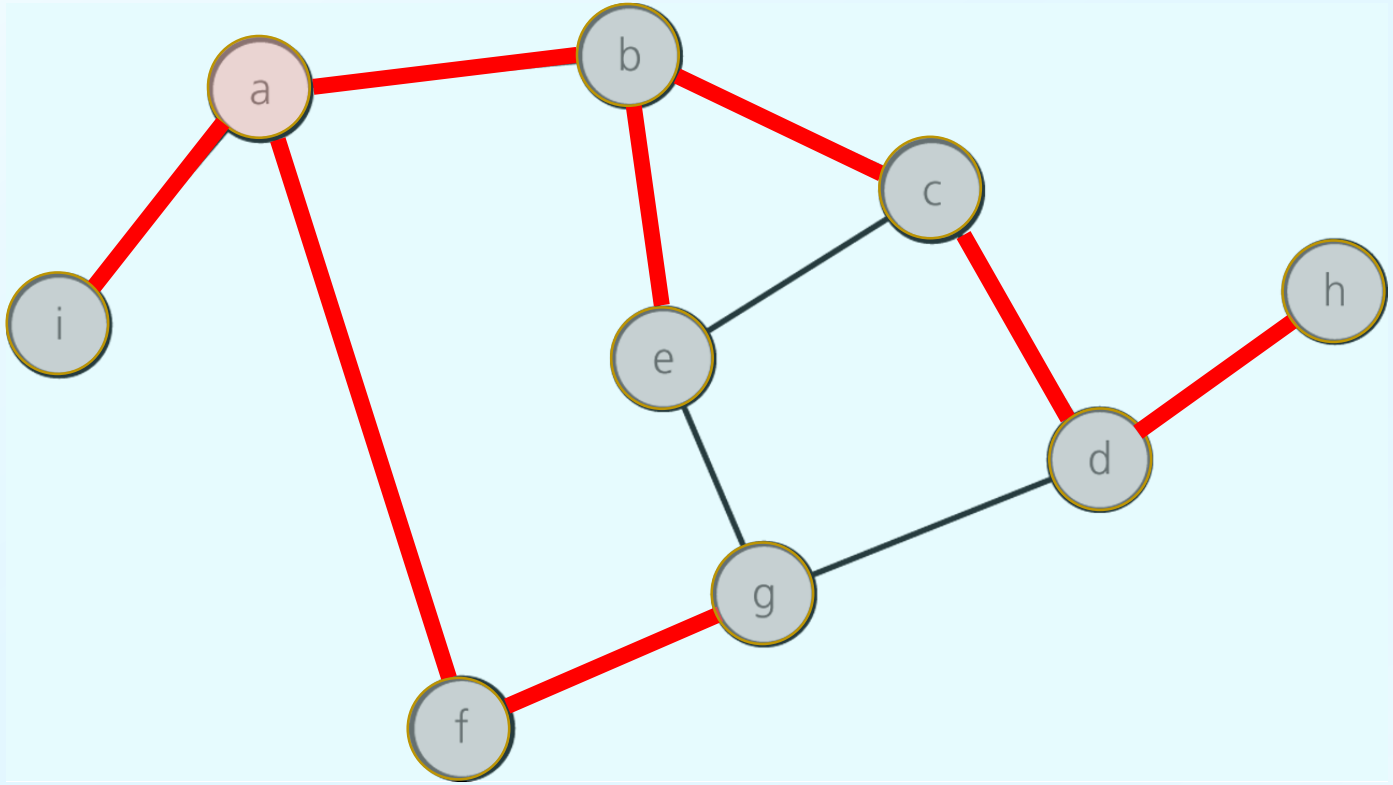
```
BFS(v):  ◄ 모든 정점은 초반에 UNVISITED로 마크됨
        queue.enqueue(v)
        mark[v] ← VISITED
        while (!queue.isEmpty())
                w ← queue.dequeue()
                for each unvisited vertex u adjacent to w
                        queue.enqueue(u)
                        mark[u] ← VISITED
```

큐를 이용한다.
기본 원리: 정점을 첫 방문할 때 큐에 삽입하고,
큐에서 정점을 하나씩 빼면서 거기서 연결된 정점들을 큐에 삽입한다.

BFS의 예

Animation

시작 정점: a



큐 | a b f i c e g d h 완료!

깊이 우선 탐색Depth-First Search

❶ DFS(*v*):

Mark *v* as visited

for (each unvisited vertex *u* adjacent to *v*)

❷ DFS(*u*)

❸ (quit, 뒷 페이지 설명을 위한 표식)

스택을 이용한다.
기본 원리: 정점을 첫 방문할 때 스택에 삽입하고, 더 이상 할 일이 없을 때 스택에서 삭제한다.
재귀 알고리즘은 암묵적으로 스택을 이용함.

비재귀적 버전

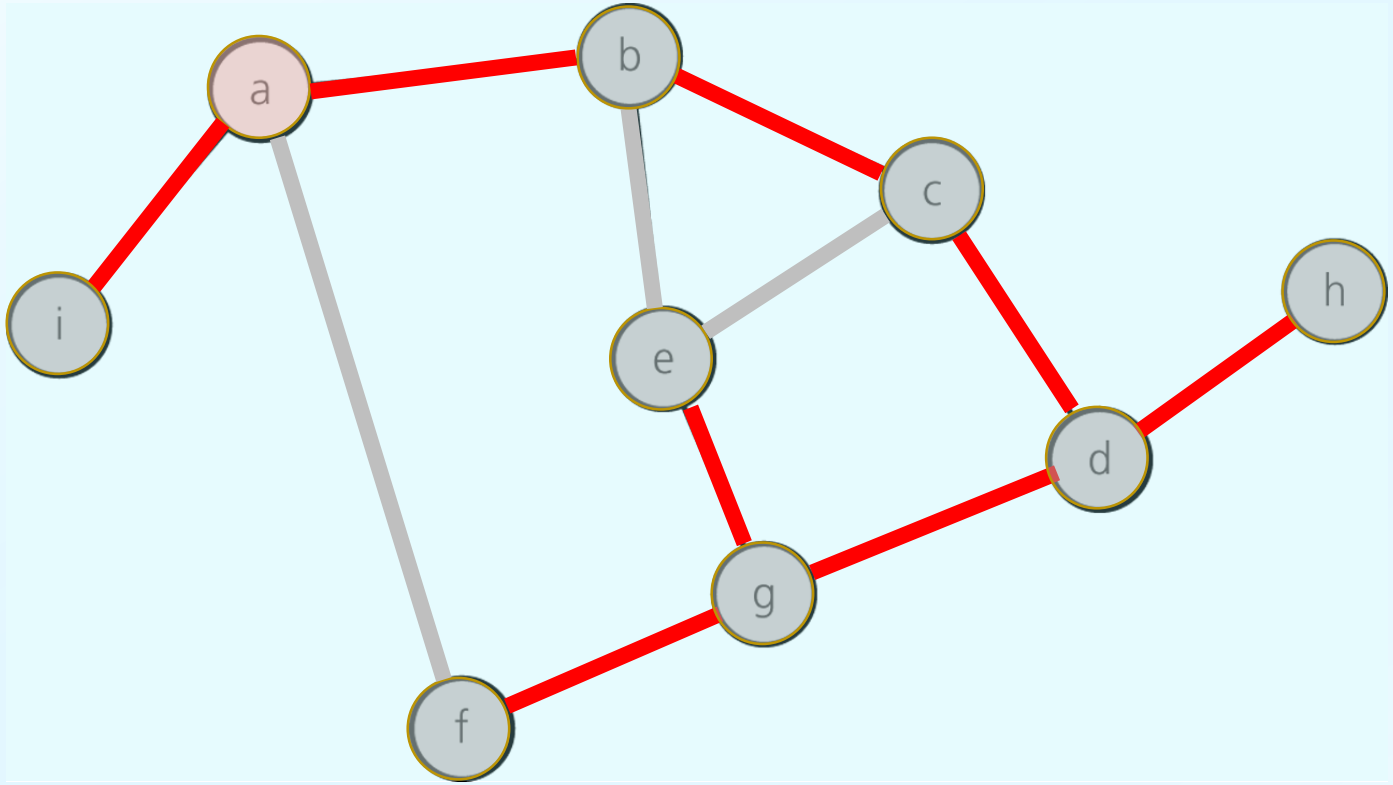
```
DFS(v): ◄ 모든 정점은 초반에 UNVISITED로 마크됨
        stack.push(v) ❶
        mark[v] ← VISITED
        while (!stack.isEmpty())
                if (no unvisited vertices are adjacent to the stack-top vertex)
                        stack.pop() ❸  ◄ backtracking
                else
                        Select an unvisited vertex w adjacent to the stack-top vertex
                        stack.push(w) ❷
                        mark[w] ← VISITED
```

스택을 명시적으로 이용한다.

DFS의 예

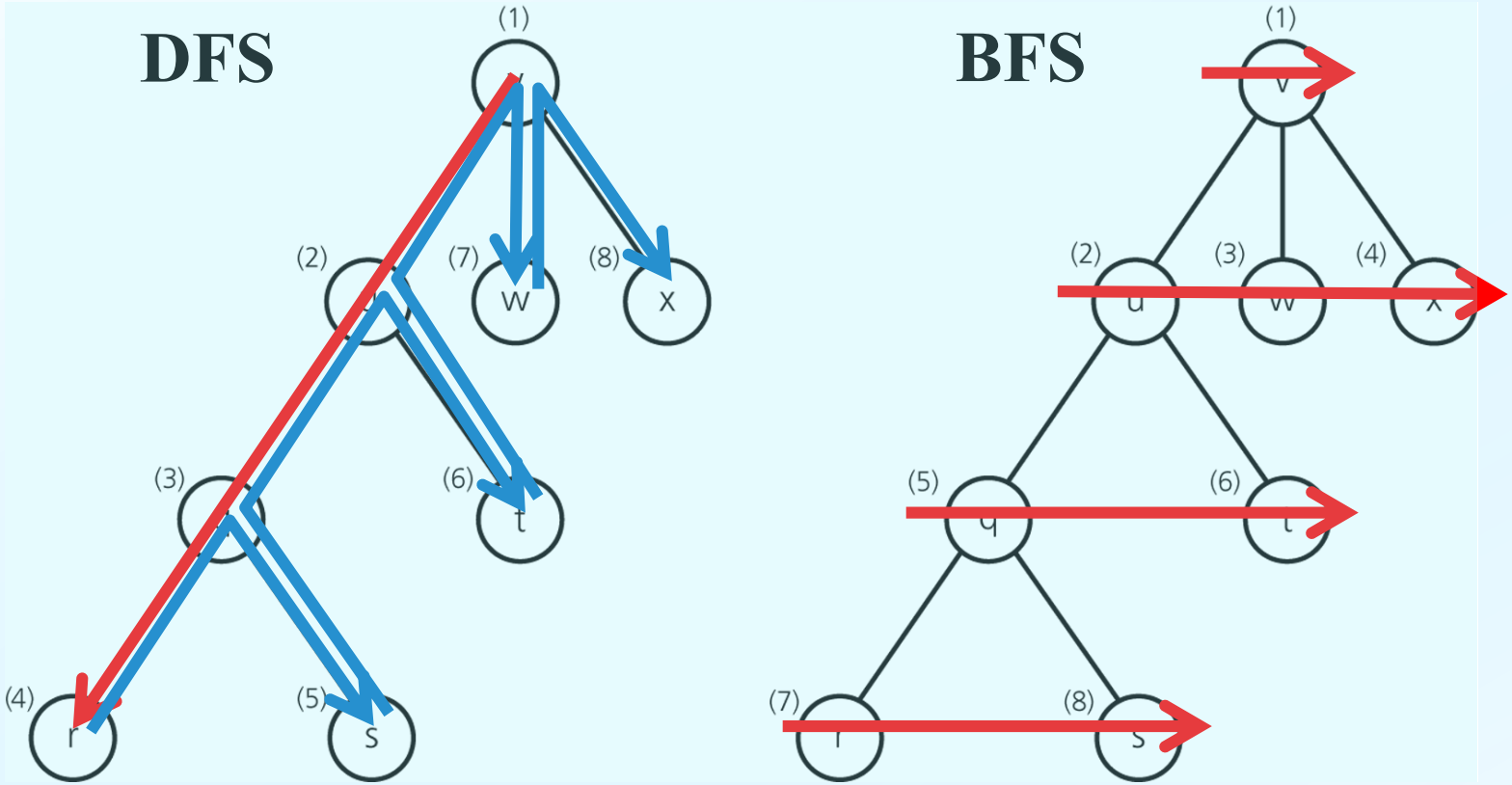
Animation

시작 정점: a



스택

동일한 트리를 각각 DFS/BFS로 방문하기



Animation

4. 최소 신장 트리

신장 트리Spanning Tree

싸이클: 출발 정점과 끝 정점이 동일한 경로

연결 그래프: 모든 정점들 간에 간선을 따라 서로 다다를 수 있는 무향 그래프

조건

- 무향 연결 그래프Undirected Connected Graph

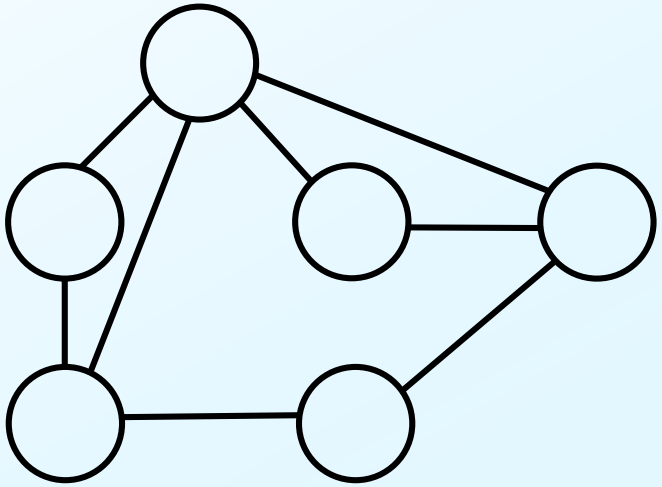
트리Tree

- 싸이클Cycle이 없는 연결 그래프
- n개의 정점으로 된 트리는 항상 n-1개의 간선을 갖는다
- 8장과 10장은 트리의 특수한 케이스(루트가 있는 트리)

그래프 G의 신장 트리

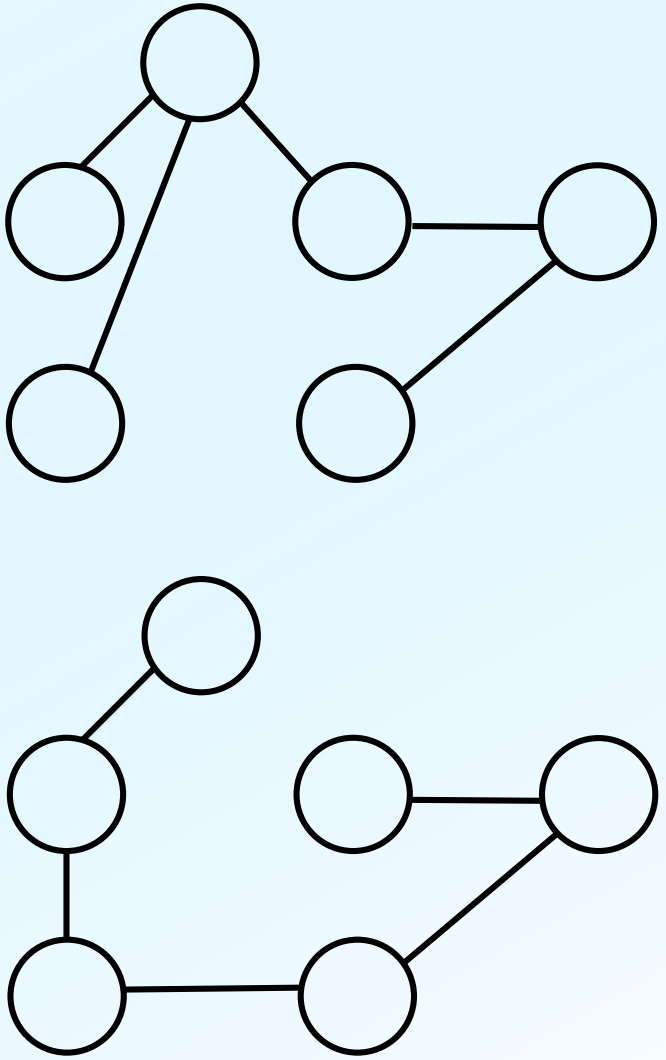
- 그래프 G에서 싸이클을 제거한 부분 그래프
- 그래프 G에서 트리가 되도록 최소한의 (n-1개) 간선만 남긴 것

그래프와 신장 트리의 예



그래프 G

G의 신장 트리의 두 예



DFS 신장 트리

DFS의 결과로 만들어지는 신장 트리

T는 공집합으로 시작함

```
DFSTree(v):
        mark[v] ← VISITED
        for (each unvisited vertex u adjacent to v)
                T = T ∪ {(u-v)}
                DFSTree(u)
```

✓ 만들어진 T는 DFS 신장 트리라 한다

보통은 간선을 $\{u, v\}$ 또는 (u, v)로 표현한다.
여기서는 좀 더 직관적인 $(u\text{-}v)$, $(u \rightarrow v)$로 표현한다.

BFS 신장 트리

BFS의 결과로 만들어지는 신장 트리

T는 공집합으로 시작함

```
BFSTree(v):  ◄ All vertices are marked UNVISITED in the beginning
        queue.enqueue(v)
        mark[v] ← VISITED
        while (!queue.isEmpty())
                w ← queue.dequeue()
                for each vertex u adjacent to w
                        queue.enqueue(u)
                        mark[u] ← VISITED
                        T = T ∪ {(u-v)}
```

✓ 만들어진 T는 BFS 신장 트리라 한다

최소 신장 트리 Minimum Spanning Trees

조건

- 가중 무향 그래프

신장 트리의 비용 Cost of a Spanning Tree

- 신장 트리를 구성하는 간선 가중치의 합

최소 신장 트리

- 비용을 최소로 하는 신장 트리

프림 알고리즘Prim Algorithm

```
Prim(G, r): ◂ G = (V, E): 주어진 그래프, r: 시작 정점
    S ← {r}   ◂ S: 정점 집합
    r.cost ← 0
❶ for each u ∈ V-{r}
        u.cost ← w_ru
❷ while (S≠V)   ◂ n-1회 순환한다
    ❸ u ← deleteMin(V-S)
      S ← S∪{u}
    ❹ for each v ∈ u.adj   ◂ u.adj: 정점 u에 인접한 정점 집합
        ❺ if (v ∈ V-S and w_uv < v.cost)
            ❻ v.cost ← w_uv
            ❼ v.tree ← u

deleteMin(Q):
    집합 Q에서 u.cost값이 가장 작은 정점 u를 삭제하면서 리턴한다
```

* 그리디 알고리즘Greedy Algorithm:
  우선 눈앞의 이익을 최대화하는
  선택을 계속하는 알고리즘을 총칭함

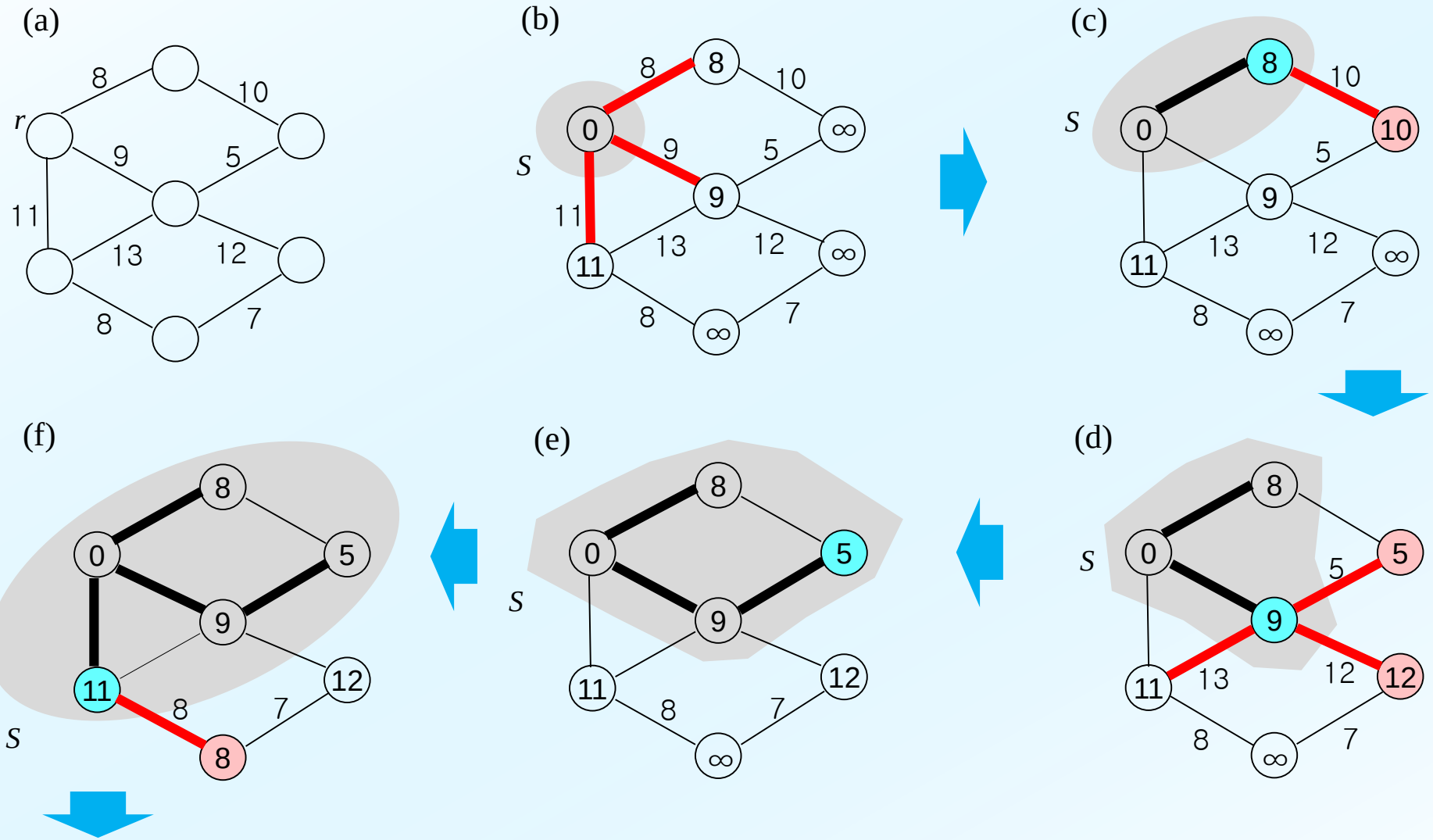
- ✓ Prim algorithm은 greedy algorithm의 예
- ✓ Greedy algorithm으로 최적해가 보장되는 드문 예

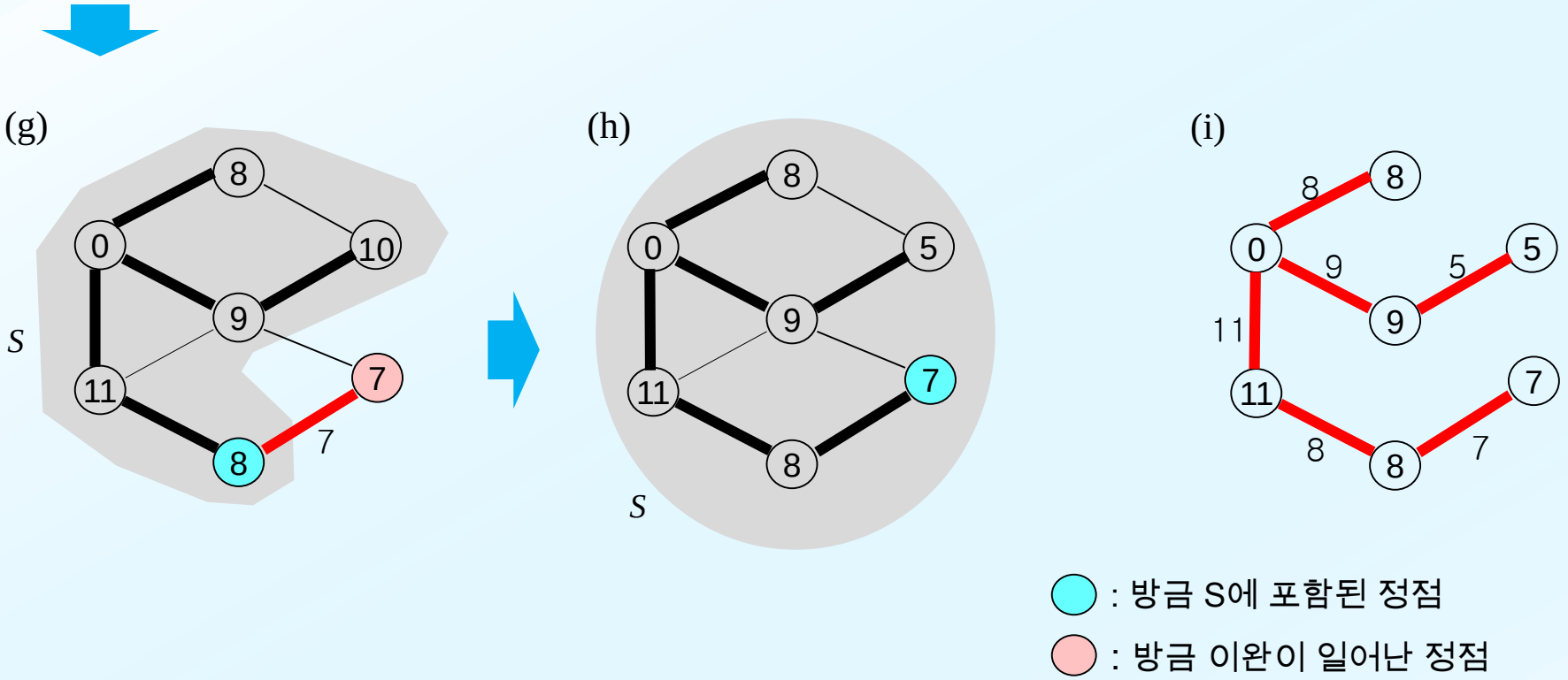
프림 알고리즘의 직관적 버전

T는 공집합으로 시작함

```
Prim(v):
        Mark v as visited
        while (there is at least an unvisited vertex)
                Find a least-cost edge (x-u) from a visited vertex x
                                                to an unvisited vertex u
                Mark u as visited
                T = T ∪ {(x-v)}
```

프림 알고리즘의 작동 예


(a)
r
8
10
9
5
11
13
12
8
7
(b)
S
8
8
10
0
9
5
∞
11
13
12
∞
8
∞
7
(c)
S
8
10
0
10
5
9
11
13
12
∞
8
∞
7
(d)
S
8
0
5
5
9
11
13
12
12
8
∞
7
(e)
S
8
0
5
9
11
12
8
∞
7
(f)
S
8
0
5
9
11
12
8
7
8


(g)
8
0
10
9
S
11
7
8
7
(h)
8
0
5
9
11
7
8
S
(i)
8
8
0
9
5
5
11
9
11
7
8
8
7
: 방금 S에 포함된 정점
: 방금 이완이 일어난 정점

크루스칼 알고리즘Kruscal Algorithm

Kruskal(G):

T ← Φ ◀ T: 신장 트리

❶ 각각 단 하나의 정점만으로 이루어진 n개의 집합을 초기화한다

❷ 모든 간선을 가중치의 크기순으로 정렬하여 배열 A[0...|E|-1]에 저장한다

❸ **while** (T의 간선수 < n-1)

❹ A에서 최소 비용의 간선 (u-v)를 제거한다

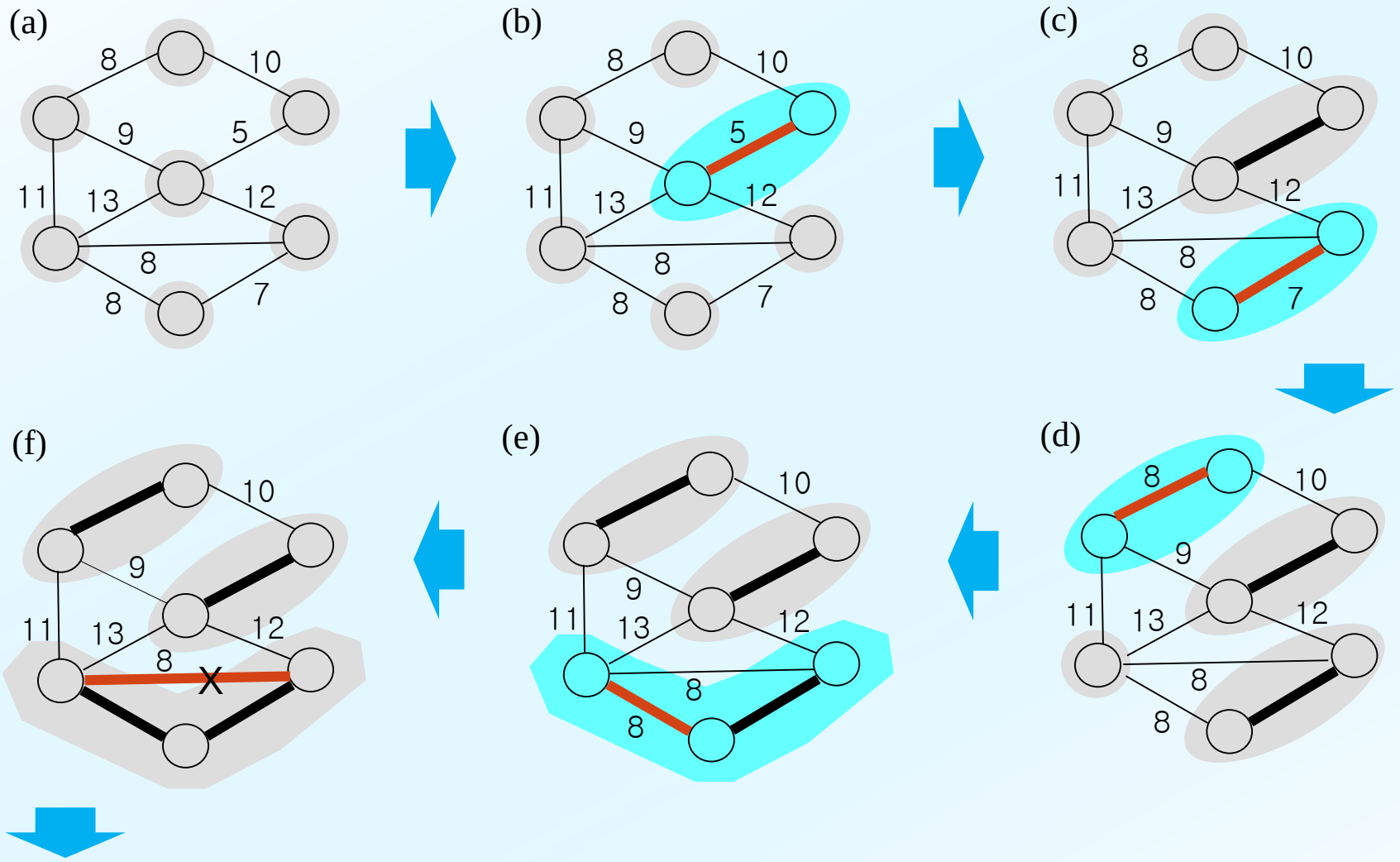
❺ **if** (정점 u와 v가 다른 집합에 속함)

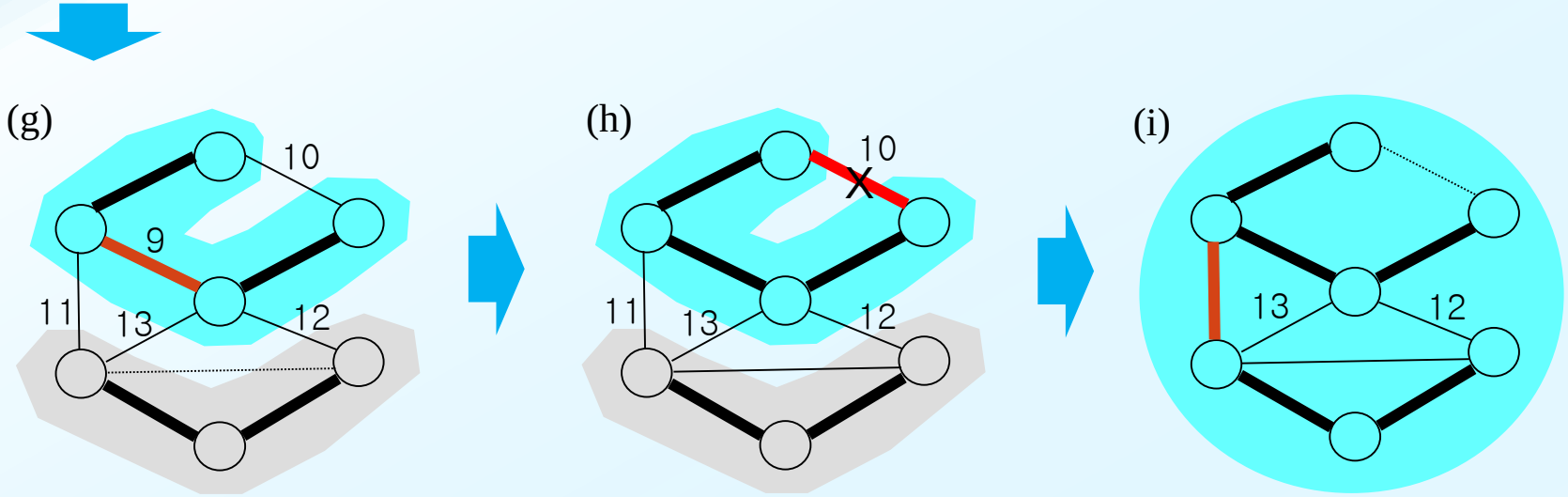
❻ T ← T∪{(u-v)} ◀ 간선 (u-v)를 더함

❼ 정점 u와 v가 속한 두 집합을 하나로 합친다

- ✓ Kruscal algorithm도 greedy algorithm의 예
- ✓ Greedy algorithm으로 최적해가 보장되는 드문 예

크루스칼 알고리즘의 작동 예

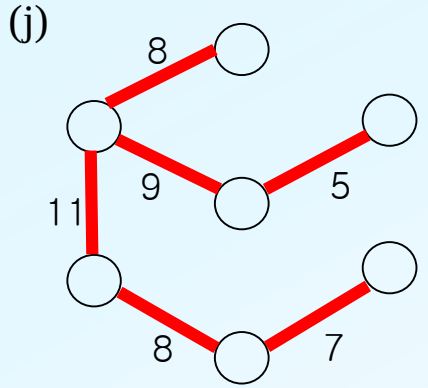




: 방금 고려한 간선

: 성공적으로 더해진 간선

X : 더하는 데 실패한 간선



5. 위상 정렬

위상 정렬Topological Sorting

조건

- 싸이클이 없는 방향 그래프

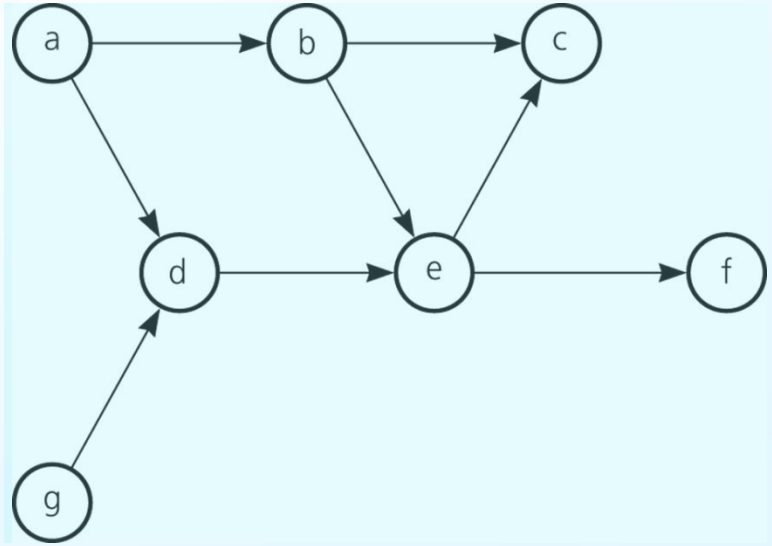
위상 순서

- 간선 $(x \rightarrow y)$가 존재하면 정점 x는 정점 y에 앞선다
- 대개 한 방향 그래프에는 서로 다른 위상 순서가 여럿 존재한다

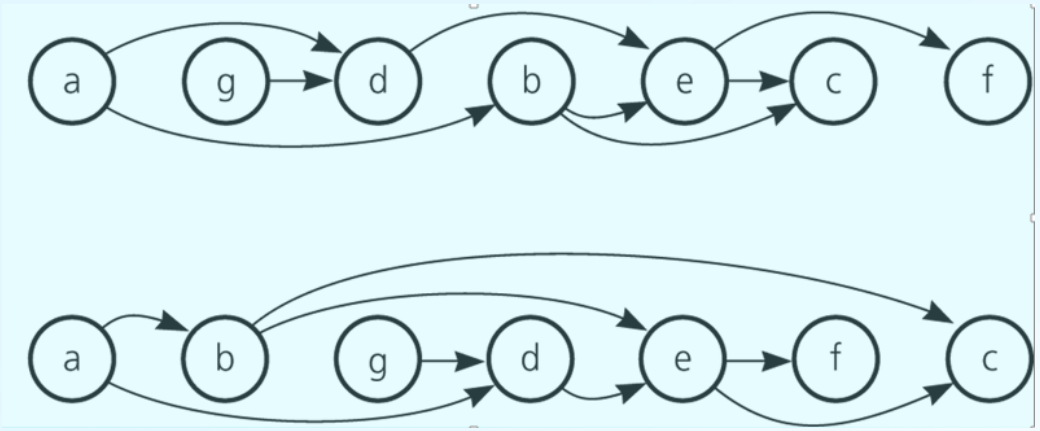
위상 정렬Topological Sorting

- 주어진 방향 그래프 G의 위상 순서 중 하나를 찾는다

A Graph and Two Examples of Topological Order



A graph and examples of topological orders



알고리즘 topologicalSort()

topologicalSort(G):

for $i \leftarrow 0$ **to** n-1

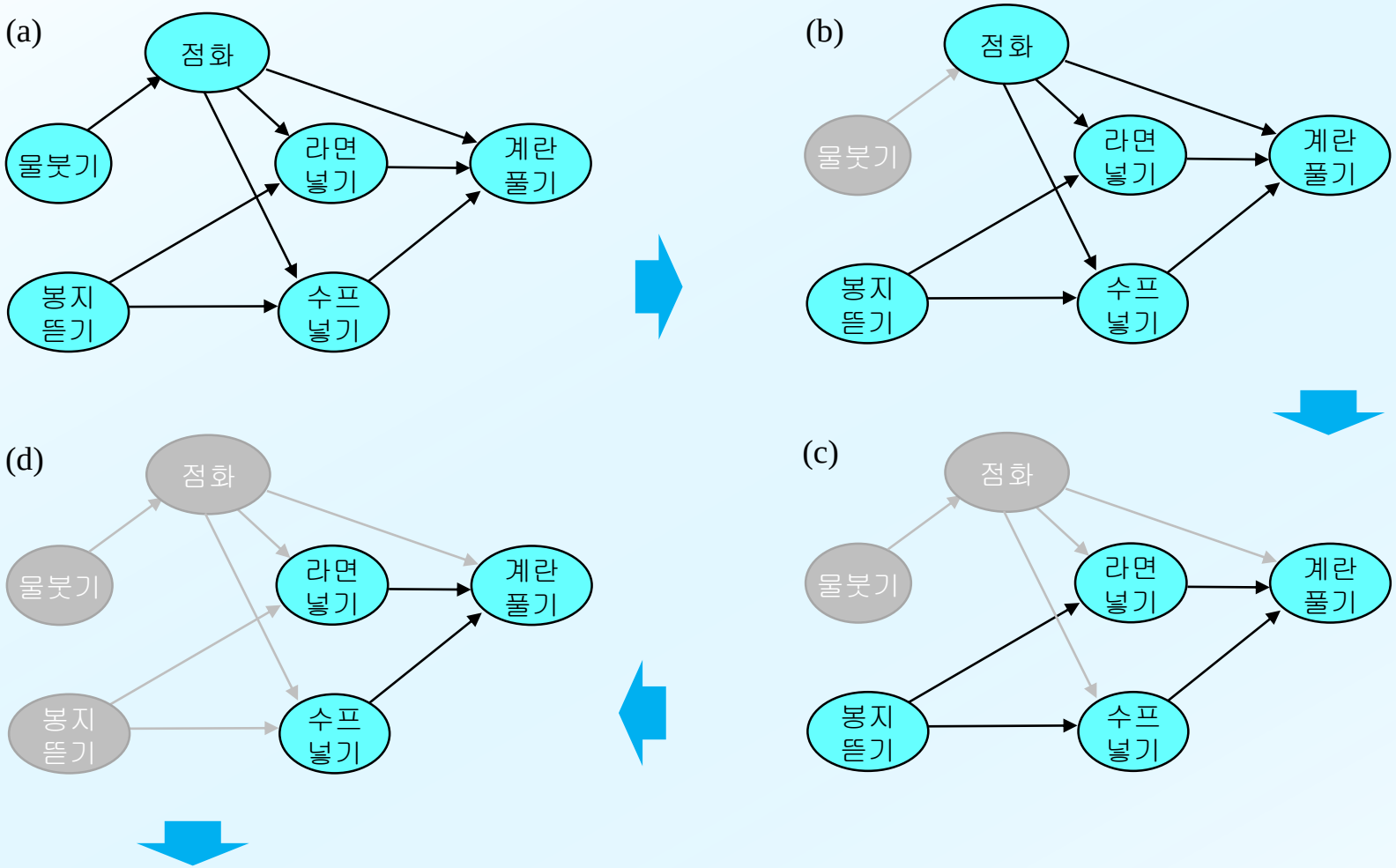
❶ 진입 간선이 없는 정점 u를 선택한다

A[i] ← u

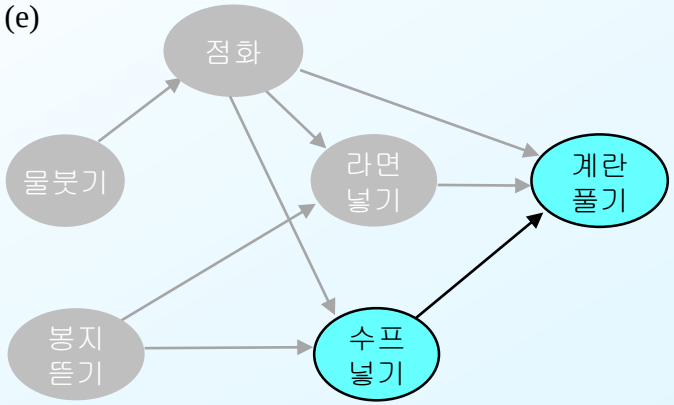
❷ 정점 u와 u의 진출 간선들을 모두 제거한다

◀ 이 시점에 배열 A[0...n-1]에는 정점들이 위상 정렬된 상태다

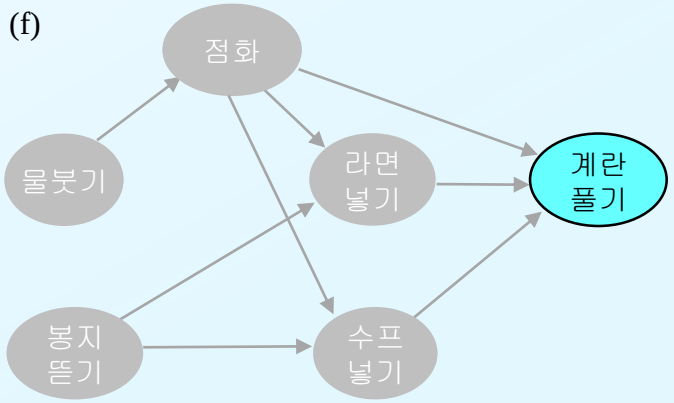
위상 정렬의 예


(a)
점화
물붓기
라면
넣기
계란
풀기
봉지
뜯기
수프
넣기
(b)
점화
물붓기
라면
넣기
계란
풀기
봉지
뜯기
수프
넣기
(c)
점화
물붓기
라면
넣기
계란
풀기
봉지
뜯기
수프
넣기
(d)
점화
물붓기
라면
넣기
계란
풀기
봉지
뜯기
수프
넣기

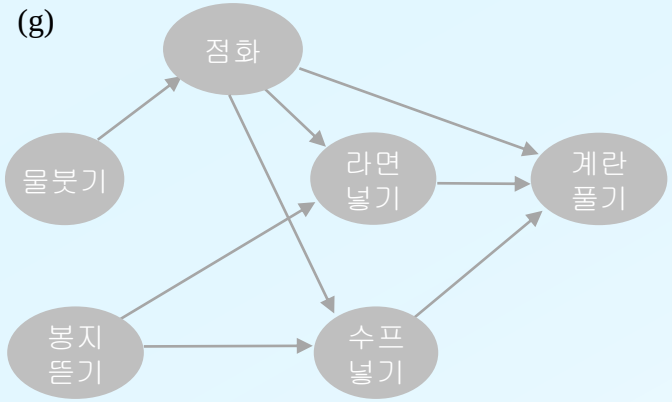
(e)


점화
물붓기
라면
넣기
계란
풀기
봉지
뜯기
수프
넣기

(f)


점화
물붓기
라면
넣기
계란
풀기
봉지
뜯기
수프
넣기

(g)


점화
물붓기
라면
넣기
계란
풀기
봉지
뜯기
수프
넣기

6. 최단 경로

최단 경로 Shortest Path

조건

- 가중 방향 그래프
- 무향 그래프도 방향 그래프로 해석할 수 있다
  - 무향 간선 $(u\text{-}v)$를 방향 간선 $(u \rightarrow v)$와 $(v \rightarrow u)$로 볼 수 있다

두 정점 사이의 최단 경로

- 경로에 있는 간선의 가중치 합을 최소로 하는 경로

간선 가중치 조건이 다르다

음의 싸이클이 있으면 모든 최단 경로 문제가 성립하지 않는다

- 다익스트라 알고리즘: 모든 간선 가중치가 음이 아니다
- 벨만-포드 알고리즘: 음의 가중치를 허용하되 음의 싸이클은 허용하지 않는다

목표: 출발 정점에서 모든 정점에 이르는 최단 경로를 구한다

다익스트라 알고리즘Dijkstra Algorithm

```
Dijkstra(G, r):   ◂ r: 출발 정점
        S ← {r}
        r.cost ← 0
        for each u ∈ V-{r}
                u.cost ← w_{r,u}
        while (S ≠ V)
                u ← deleteMin(V-S)
                S ← S ∪ {u}
                for each v ∈ u.adj   ◂ u.adj: 정점 u에 인접한 정점들의 집합
                        if (v ∈V-S and u.cost + w_{v,u} < v.cost)
                                v.cost ← u.cost + w_{u,v}   ← Relaxation!
                                v.prev ← u
```

* *v*.prev : 최단 경로를 만들 수 있는 연결 정보

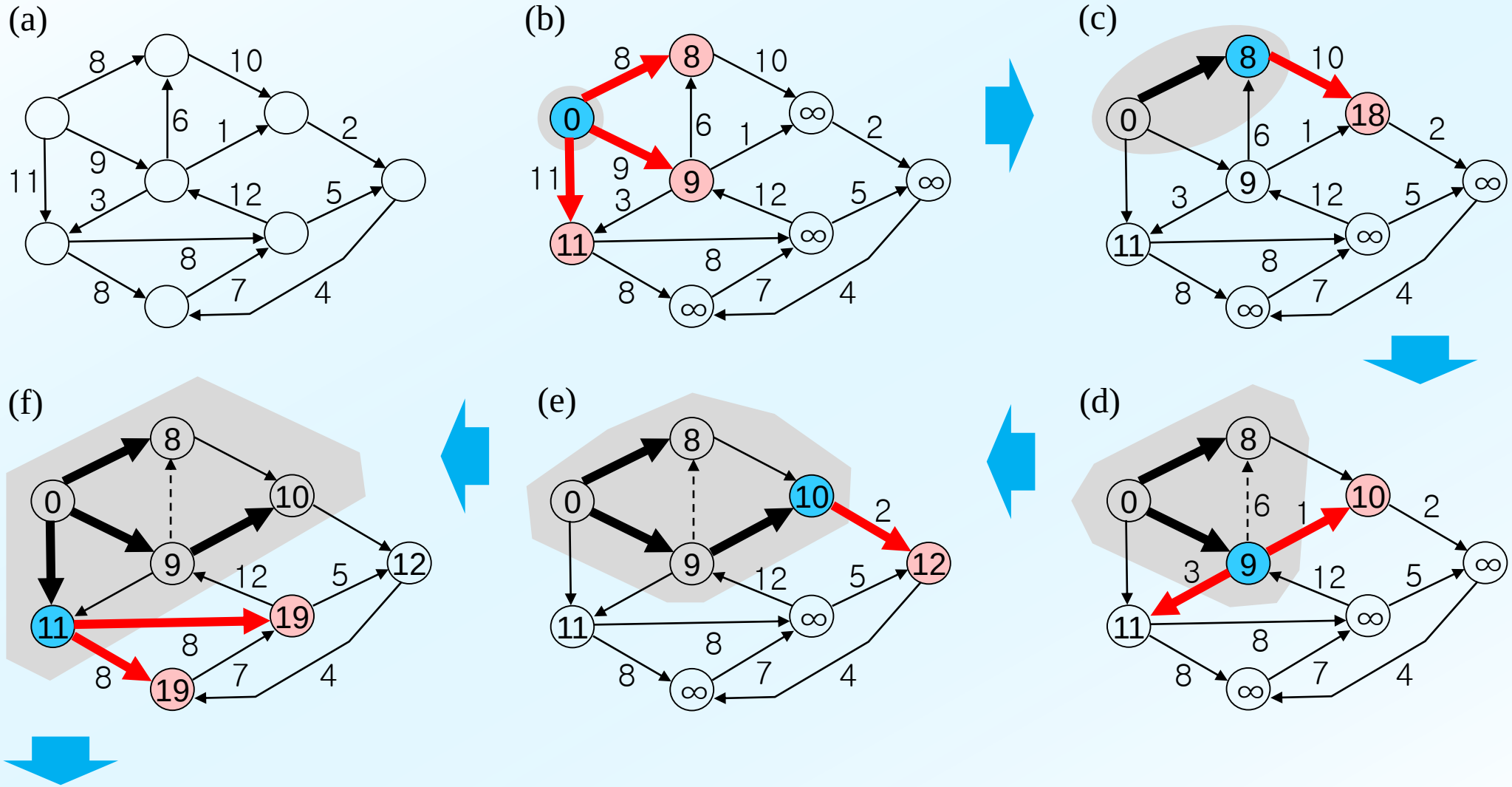
- ✓ 다익스트라 알고리즘은 그리디 알고리즘의 예
- ✓ 그리디 알고리즘으로 최적해가 보장되는 드문 예

프림 알고리즘과 다익스트라 알고리즘은 로직이 거의 같다

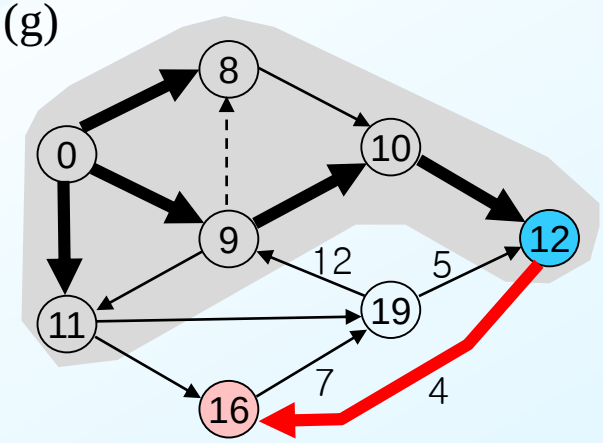
```
Prim(G, r): ◂ G = (V, E): 주어진 그래프, r: 시작 정점
    S ← {r}  ◂ S: 정점 집합
    r.cost ← 0
❶ for each u ∈ V-{r}
        u.cost ← w_ru
❷ while (S≠V)  ◂ n-1회 순환한다
    ❸ u ← deleteMin(V-S)
      S ← S ∪ {u}
    ❹ for each v ∈ u.adj  ◂ u.adj: 정점 u에 인접한 정점 집합
        ❺ if (v ∈ V-S and w_uv < v.cost)
            ❻ v.cost ← w_uv

deleteMin(Q):
    집합 Q에서 u.cost값이 가장 작은 정점 u를 삭제하면서 리턴한다
```

```
Dijkstra(G, r):  ◂ r: 출발 정점
    S ← {r}
    r.cost ← 0
    for each u ∈ V-{r}
        u.cost ← w_r,u
    while (S ≠ V)
        u ← deleteMin(V-S)
        S ← S ∪ {u}
        for each v ∈ u.adj  ◂ u.adj: 정점 u에 인접한 정점들의 집합
            if (v ∈V-S and u.cost + w_v,u< v.cost)
                v.cost ← u.cost + w_u,v
```

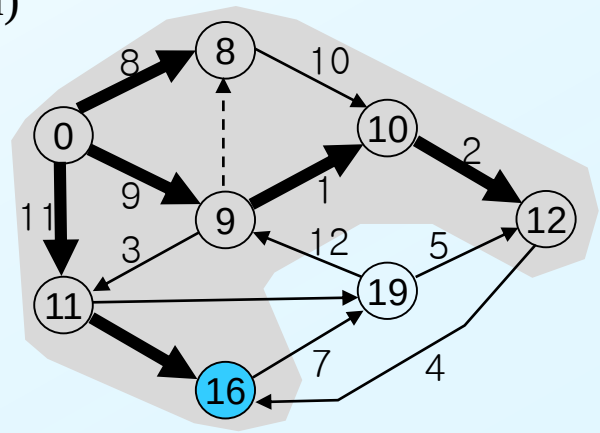
다익스트라 알고리즘 적용 예



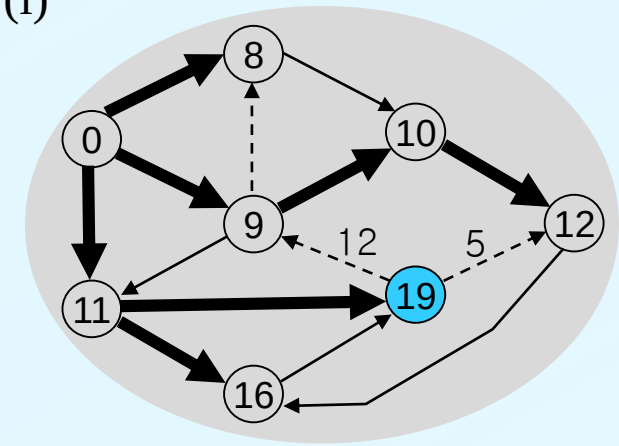
(g)


8
0
10
9
12
12
5
19
11
7
4
16

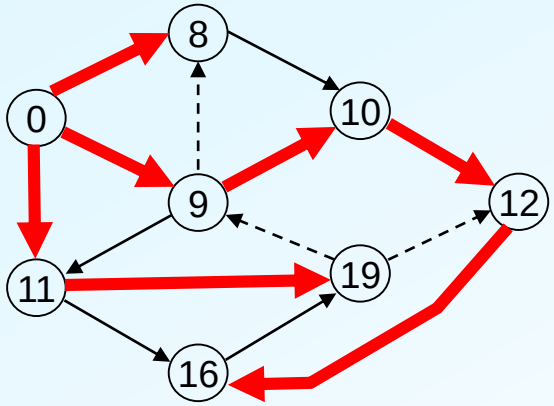
(h)


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

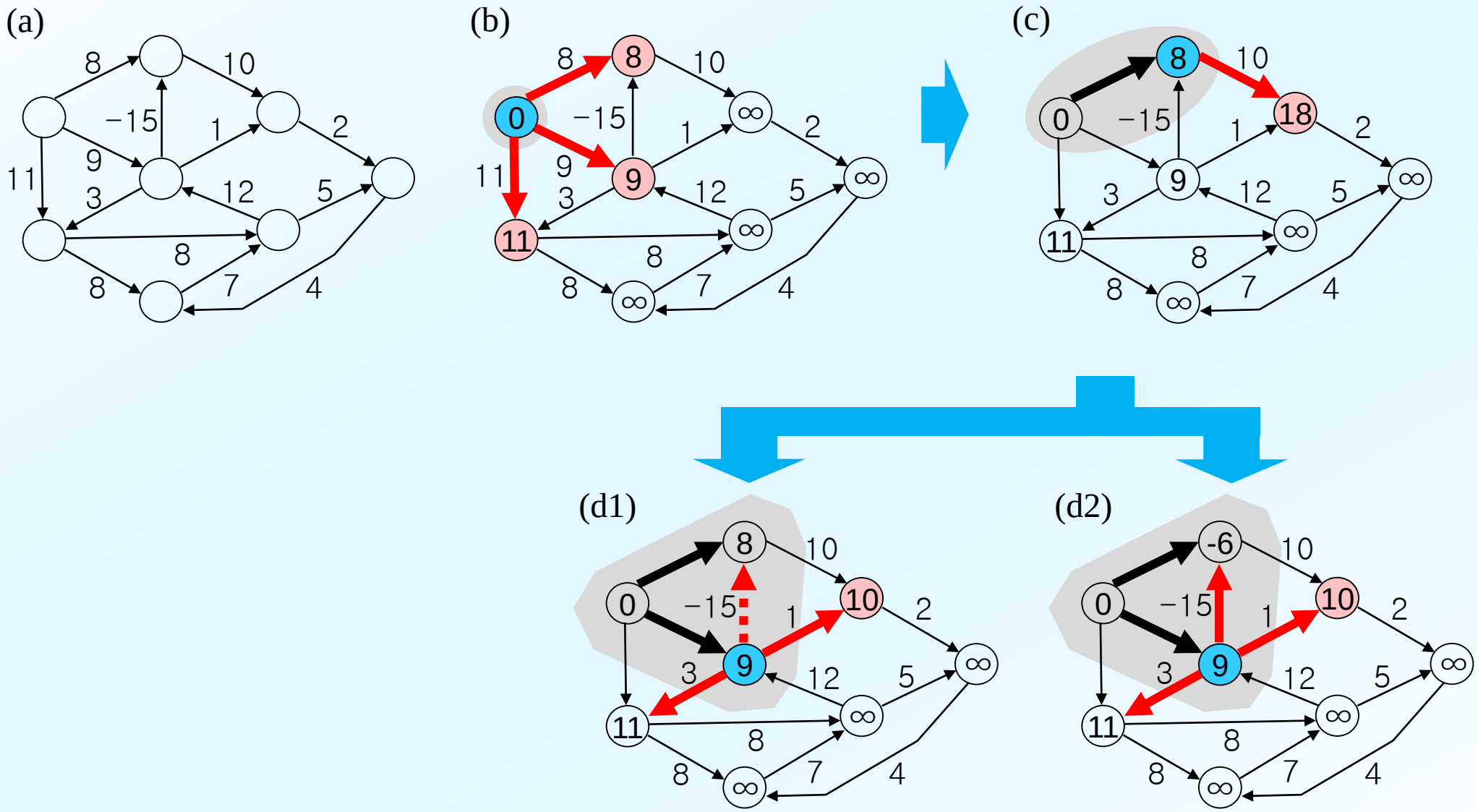
(i)


8
0
10
9
12
12
5
19
11
16

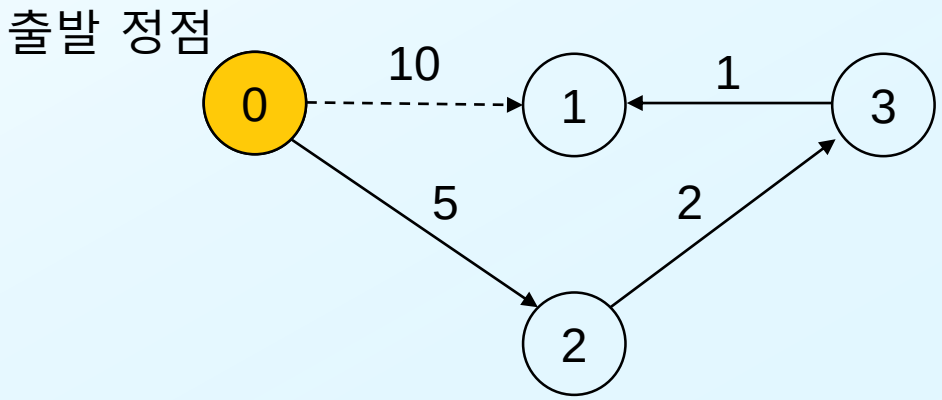
(j)


8
0
10
9
12
19
11
16

다익스트라 알고리즘이 작동하지 않는 예



이완의 예

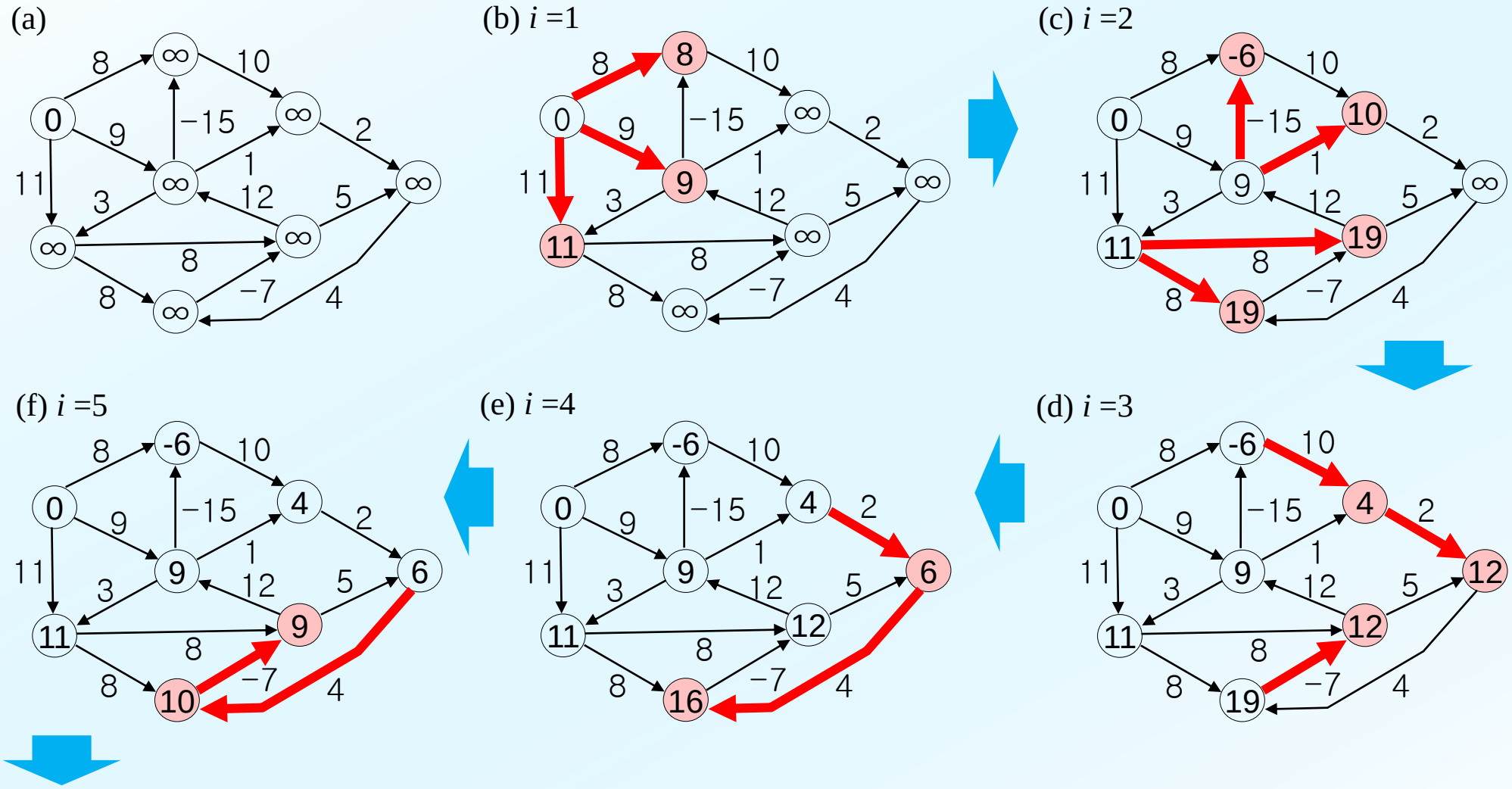


정점 1에 이르는 거리는 10으로 계산되어 시작하는데, 나중에 8로 바뀐다.

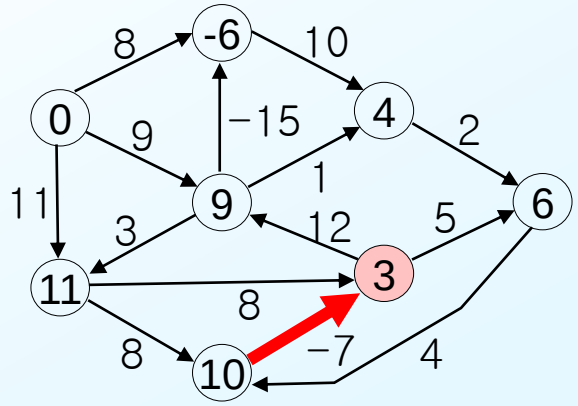
벨만-포드 알고리즘Bellman-Ford Algorithm

```
BellmanFord(G, r): ◀ G = (V, E): 주어진 그래프, r: 시작 정점
        for each u∈V
                u.cost ← ∞
        r.cost ← 0
        for i ← 1 to n-1  ◀ |V| = n: 정점의 총 수
                for each (u→v) ∈ E  ◀ (u→v): 방향 간선
                        if (u.cost + w_u,v < v.cost)
                                v.cost ← u.cost + w_u,v   ←——— Relaxation!
                                v.prev ← u
        ◀ 음의 싸이클 존재 여부 확인
        for each (u→v) ∈ E
                if (u.cost + w_u,v < v.cost) output "해 없음: 음의 싸이클"
```

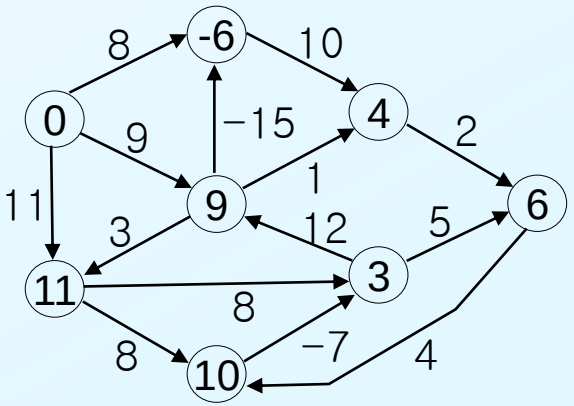
벨만-포드 알고리즘 적용 예



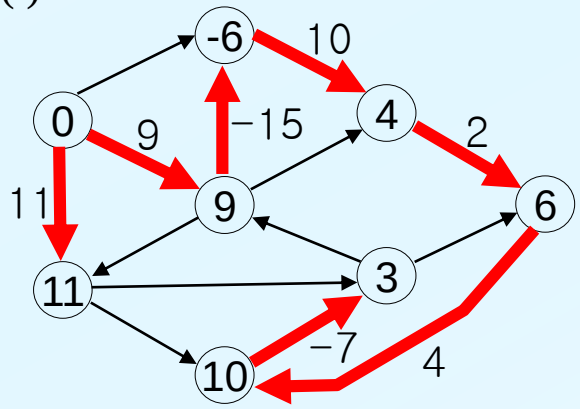
(g) i =6


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

(h) i =7

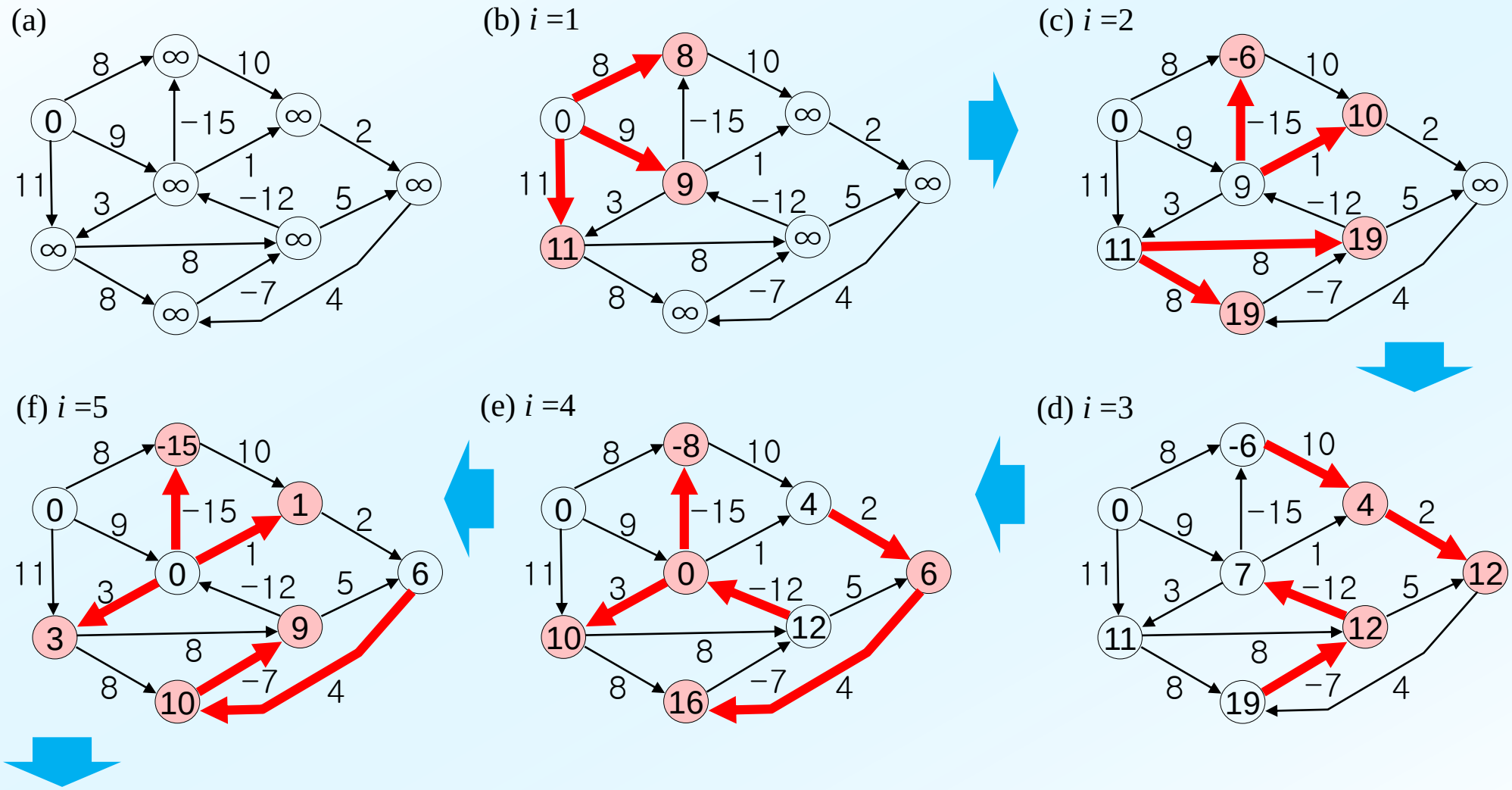

-6
0
4
9
6
11
3
10
8
10
-15
9
2
1
11
3
12
5
8
8
-7
4

(i)

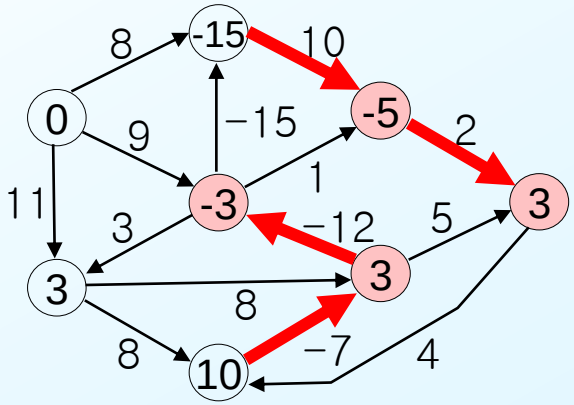

-6
0
4
9
6
11
3
10
10
-15
9
2
11
-7
4

음의 싸이클이 있는 경우

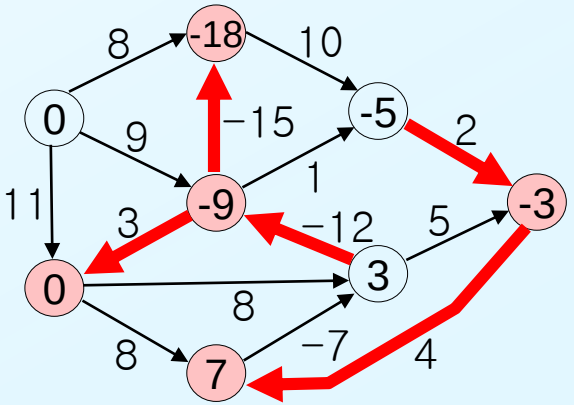
벨만-포드 알고리즘이 작동하지 않는 예



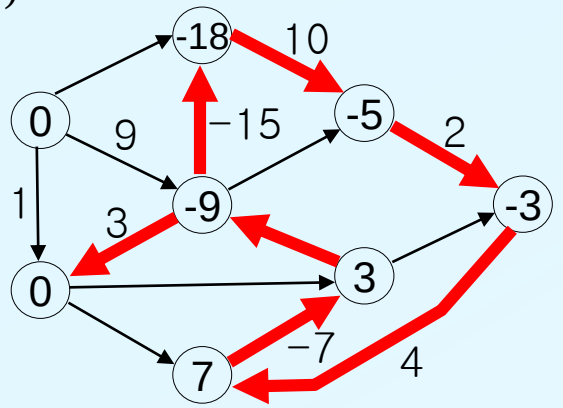
(g) i =6


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

(h) i =7


-18
10
8
0
9
−15
-5
2
1
11
-9
-3
3
−12
5
3
8
8
7
−7
4

(i)


-18
10
0
9
−15
-5
2
11
-9
-3
3
3
0
7
−7
4