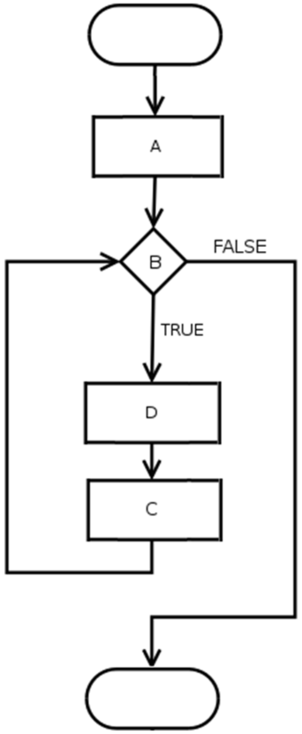
Clickseo.com

자료구조 & 알고리즘

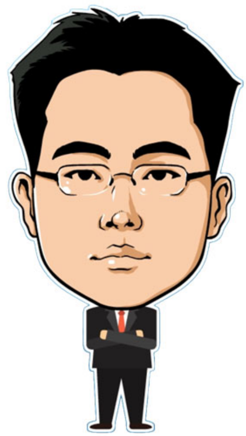
for(A;B;C)
  D;



그래프
(Graph)

Seo, Doo-Ok

Clickseo.com
clickseo@gmail.com



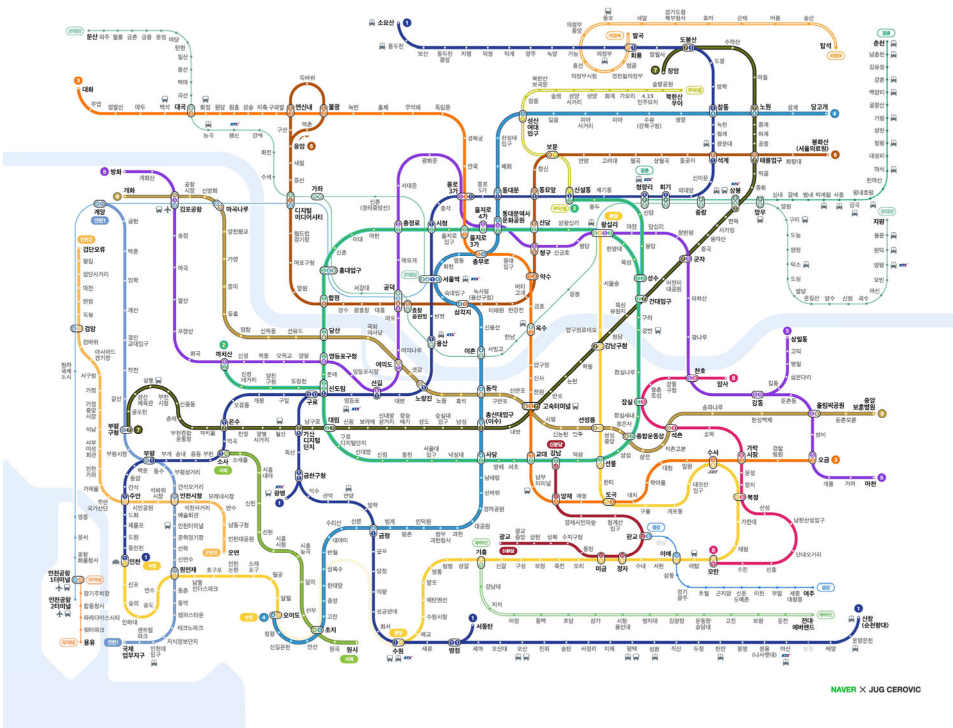
목 차

그래프의 이해

- **그래프의 이해**
  - 그래프 용어
  - 그래프 종류
  - 그래프 표현
  - 그래프 순회
- **최소 신장 트리**
- **위상 정렬**
- **최단 경로**

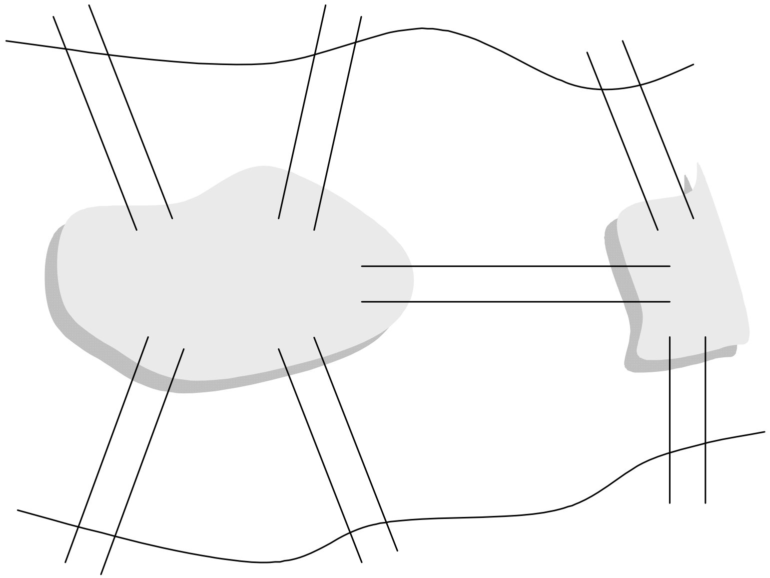
그래프 이해 (1/3)

- **그래프(Graph)**
  - 연결되어 있는 원소 간의 관계를 표현하는 자료구조
    - **그래프의 예:** 인맥 지도, 수도 배관 배수 시스템, 물질의 분자 구조



그래프 이해 (2/3)

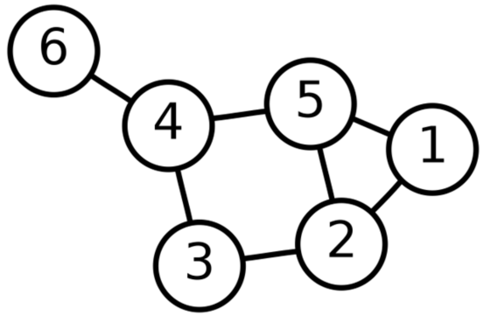
- **케인즈버그(Koenigsberg)의 다리 문제**
  - **1736년, 오일러(Euler)가 최초로 사용한 것**
    - 섬은 **정점(vertex)**으로 놓고, 다리를 **간선(edges)**으로 나타냄.
    - **각 정점의 차수가 짝수인 경우에만 임의의 정점에서 출발하여 각 간선을 단 한 번씩만 거치고 출발한 정점으로 되돌아오는 길이 있음을 보였다.**



그래프 이해 (3/3)

● 그래프 정의

- 그래프 G는 집합(Set) 두 개로 구성
  - 정점(Vertex 또는 Node)
  - 간선(Edge)



$$G = (V, E)$$

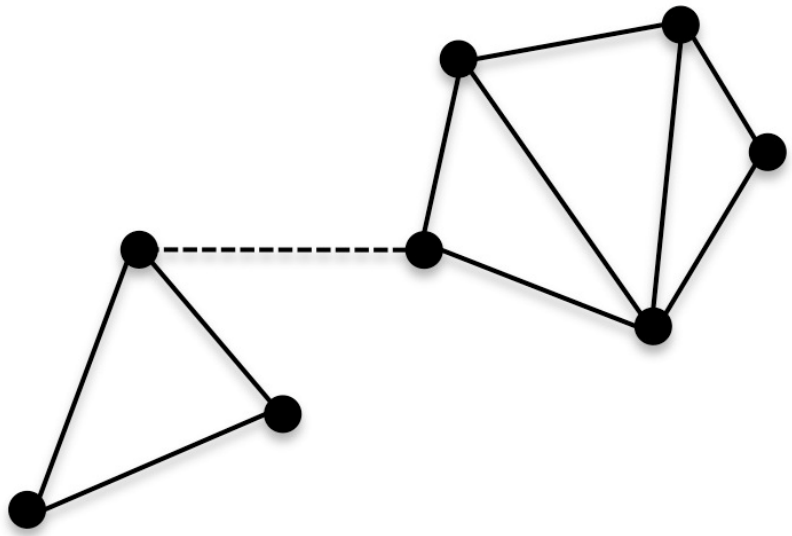
- V 는 그래프에 있는 정점들의 집합을 의미한다(대상: 대상물, 개념 등).
- E 는 정점을 연결하는 간선들의 집합을 의미한다(대상들 간의 관계).

그래프의 이해

그래프 용어

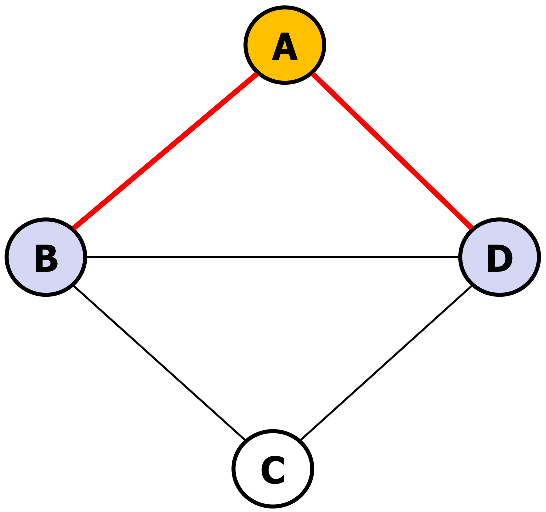
그래프 용어 (1/6)

- **연결(Connected)**
  - 그래프에서 <u>두 정점 V_i와 V_j까지의 경로가 있으면,</u> 정점 V_i와 V_j가 연결되었다고 한다.
    - **연결 그래프**(Connected Graph): 떨어져 있는 정점이 없는 그래프
    - **단절 그래프**(Disconnected Graph): 연결되지 않은 정점이 있는 그래프



그래프 용어 (2/6)

- **인접과 부속**(Adjacent and Incident)
  - 그래프에서 두 정점 V_i와 V_j가 연결되어 간선 (V_i, V_j)가 있을 때 두 정점 V_i와 V_j를 인접(adjacent)되어 있다고 하고, 간선 (V_i, V_j)는 정점 V_i와 V_j에 부속(incident)되어 있다고 한다.



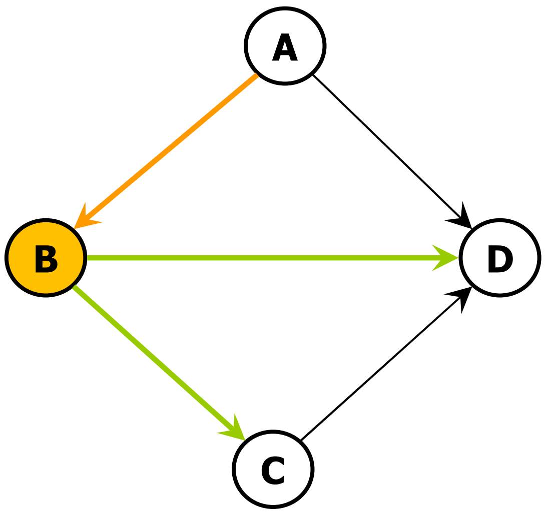
[그래프 G1]

정점 A 와 인접한 정점은 B 와 D

: 정점 A에 부속되어 있는 간선은 (A, B)와 (A, D)

그래프 용어 (3/6)

- **차수(Degree)**
  - 정점에 부속되어 있는 간선의 수
    - 유향 그래프에서는 정점에 부속된 간선의 방향에 따라서
      - 진입 차수(in-degree)
      - 진출 차수(out-degree)
    - 유향 그래프에서의 정점의 차수는 진입 차수와 진출 차수를 합한 값이다.

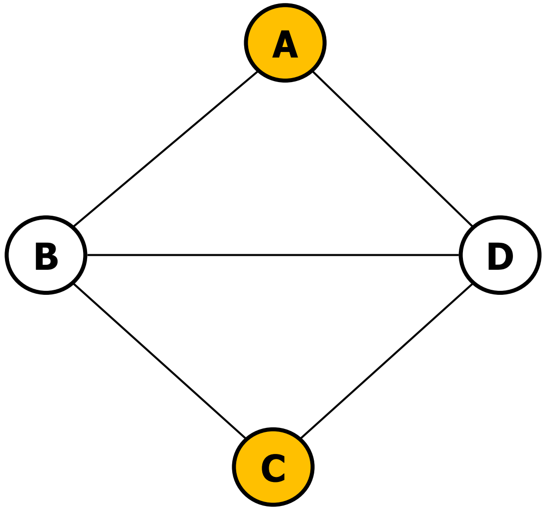


[그래프 G3]

정점 B 의 진입 차수는 1 이고, 진출 차수는 2 이다.
정점 B 의 전체 차수는 3 이다.

그래프 용어 (4/6)

- **경로**(Path)
  - 그래프에서 간선을 따라 갈 수 있는 길을 순서대로 나열 한 것



[그래프 G1]

정점 A 에서 정점 C 까지는 4 가지의 경로가 존재

A-B-C

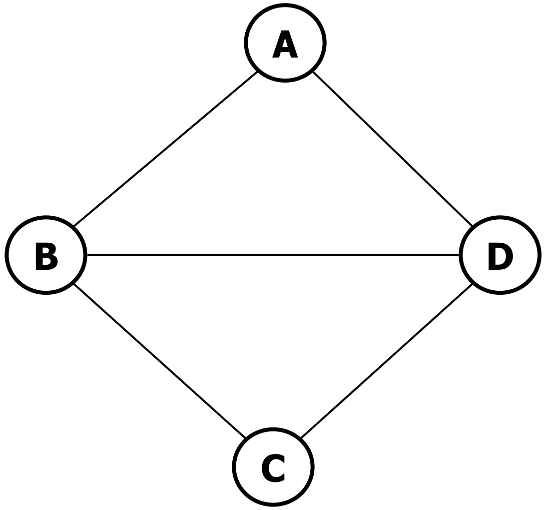
A-B-D-C

A-D-C

A-D-B-C

그래프 용어 (5/6)

- **경로:** 경로 길이
  - 경로 길이(Path Length): 경로를 구성하는 간선의 수



[그래프 G1]

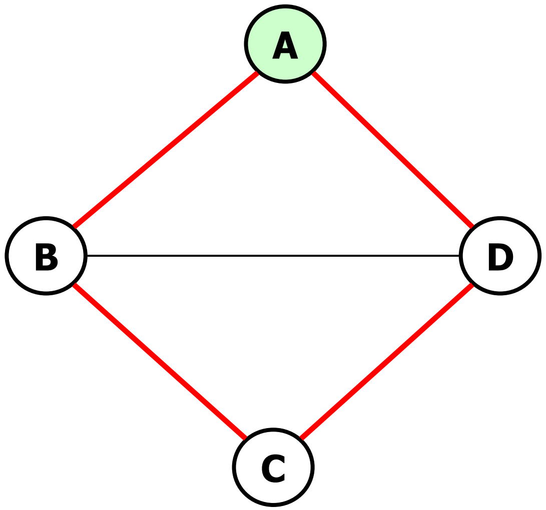
경로 A-D-B-C 는 간선 3개로 이루어진다.

(A, D)
(D, B)
(B, C)

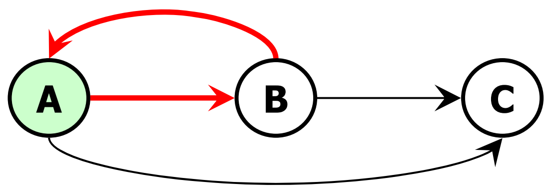
경로 길이는 3 이 된다.

그래프 용어 (6/6)

- **경로: 순환**
  - **단순 경로:** 모두 다른 정점으로 구성된 경로
  - **순환**(Cycle)
    - 단순 경로 중에서 경로의 시작 정점과 마지막 정점이 같은 경로



[그래프 G1]



[그래프 G4]

그래프 이해

그래프 종류

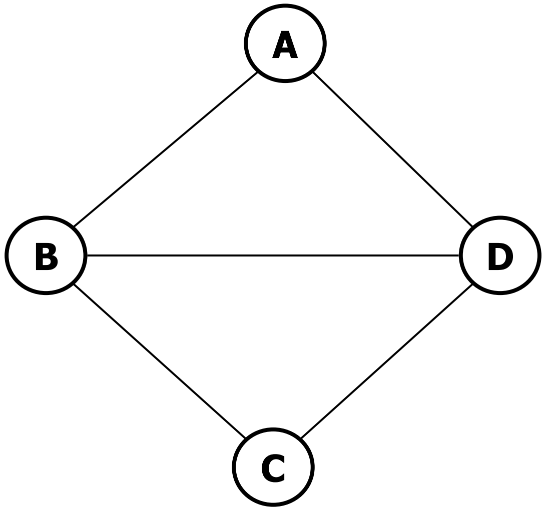
그래프 종류 (1/7)

그래프 종류

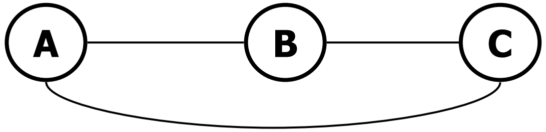
- **무향 그래프**(Undirected Graph)
  - 두 정점을 연결하는 간선에 방향성이 없는 그래프
- **유향 그래프**(Directed Graph)
  - 두 정점을 연결하는 간선에 방향성이 있는 그래프
- **가중치 그래프**(Weight Graph)
  - **두 정점을 연결하는 간선에 가중치가 할당된 그래프**
    - 가중치는 두 정점 사이의 거리 또는 지나는 시간이 될 수도 있다.
    - 또한 음수인 경우도 존재한다.
- **완전 그래프**(Complete Graph)
  - **모든 정점들 사이에 1:1로 직접 연결된 간선을 지닌 그래프**
- **부분 그래프**(Subgraph)
  - **원래의 그래프에서 일부의 정점이나 간선을 제외하여 만든 그래프**
    - 부분 그래프는 원래의 그래프에 없는 정점이나 간선을 포함하지 않는다.
- **트리(Tree): 순환이 없는 연결된 그래프**

그래프 종류 (2/7)

- **무향 그래프**(Undirected Graph)
  - 두 정점을 연결하는 간선에 방향성이 없는 그래프



[그래프 G1]



[그래프 G2]

V(G1) = { A, B, C, D }

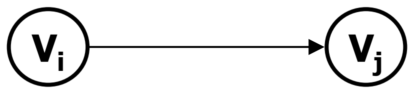
E(G1) = { (A,B), (A,D), (B,C), (B,D), (C,D) }

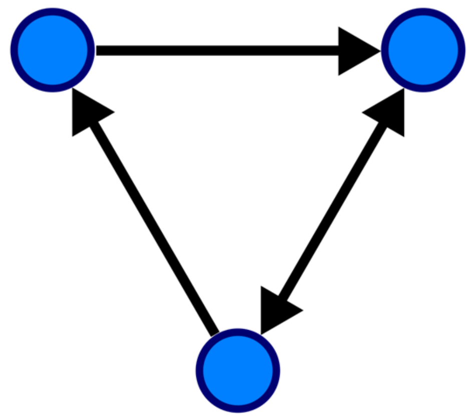
V(G2) = { A, B, C }

E(G1) = { (A,B), (A,C), (B,C)}

그래프 종류 (3/7)

- **유향 그래프**(Directed Graph)
  - 두 정점을 연결하는 간선에 <u>방향성이 있는</u> 그래프
    - 정점 V_i에서 정점 V_j를 연결하는 간선





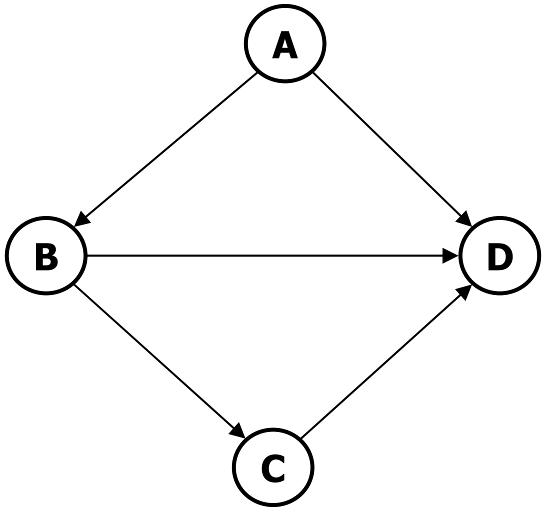
$V_i \rightarrow V_j$ 또는 $<V_i, V_j>$ 로 표현

$<V_i, V_j>$와 $<V_j, V_i>$는 서로 다른 간선이 된다.

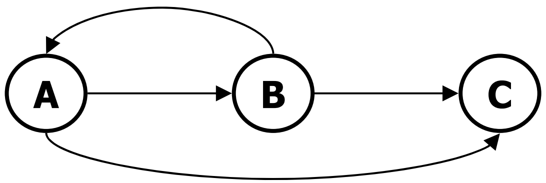
그래프 종류 (4/7)

유향 그래프

- 그래프 G3, G4



[그래프 G3]



[그래프 G4]

V(G3) = { A, B, C, D }

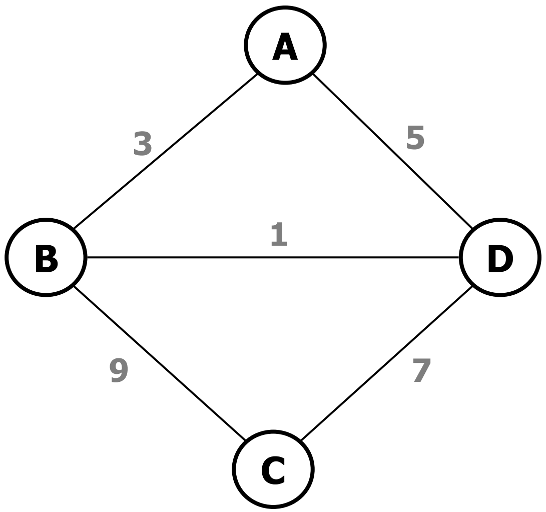
E(G3) = { <A,B>, <A,D>, <B,C>, <B,D>, <C,D> }

V(G4) = { A, B, C }

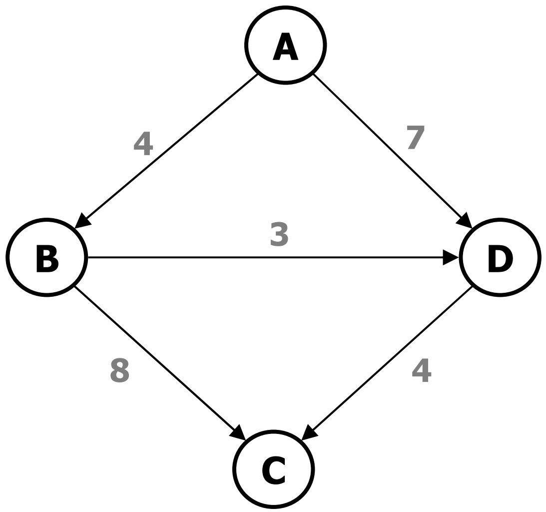
E(G4) = { <A,B>, <A,C>, <B,A>, <B,C> }

그래프 종류 (5/7)

- **가중치 그래프**(Weight Graph)
  - 두 정점을 연결하는 간선에 가중치를 할당한 그래프
    - 가중치는 두 정점 사이의 거리 또는 지나는 시간이 될 수도 있다.
    - 또한 음수인 경우도 존재한다.



[그래프 G5]

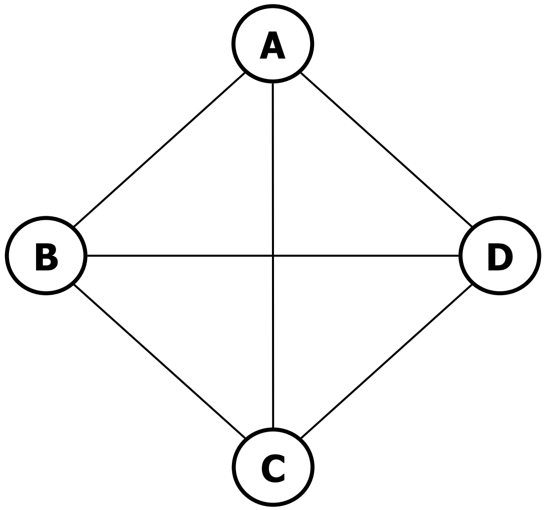


[그래프 G6]

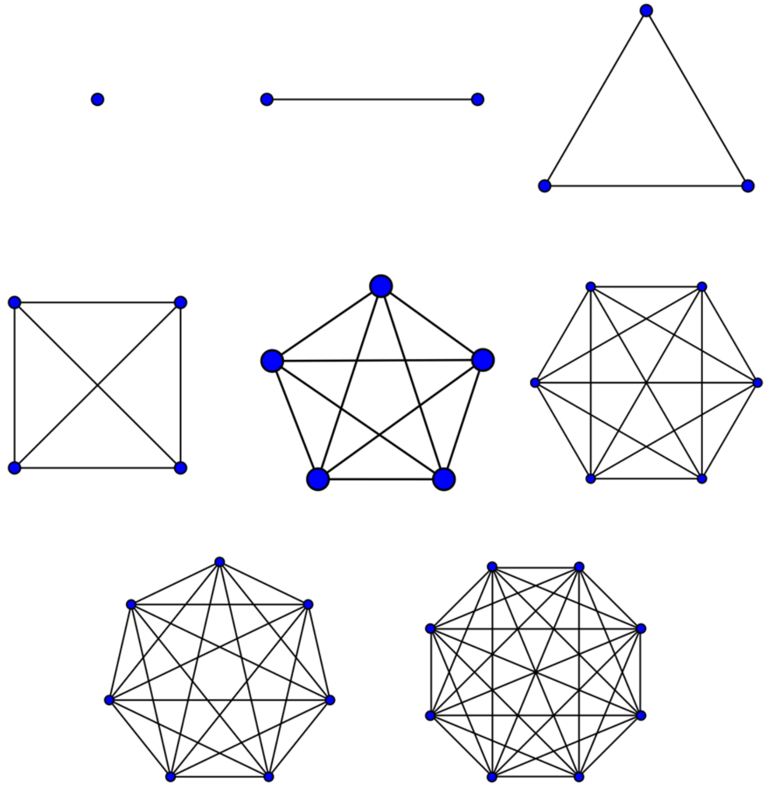
그래프 종류 (6/7)

- **완전 그래프**(Complete Graph)
  - 모든 정점들 사이에 1:1로 직접 연결된 간선을 지닌 그래프

정점이 n 개인 무방향 그래프 최대 간선 수 = $n(n-1)/2$

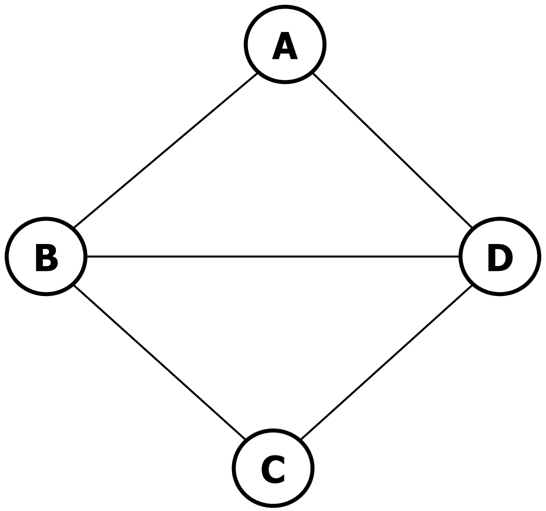


[그래프 G7]

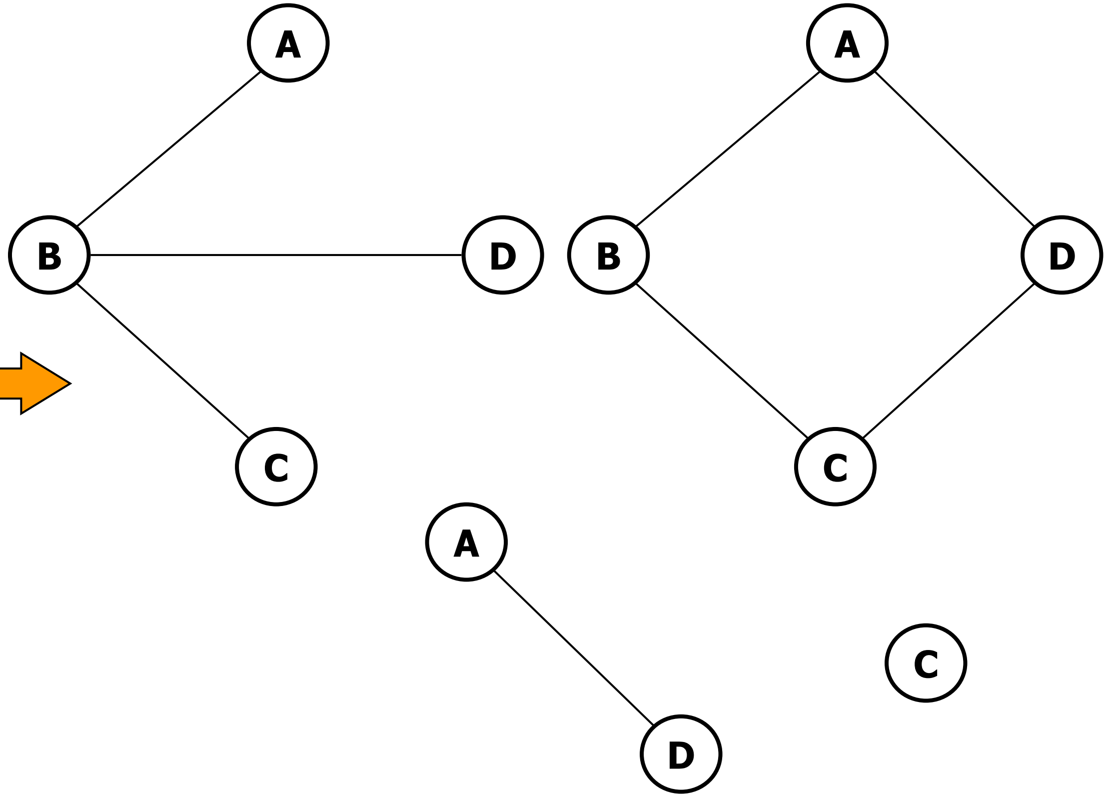


그래프 종류 (7/7)

- **부분 그래프**(subgraph)
  - 원래의 그래프에서 일부의 정점이나 간선을 제외하여 만든 그래프
    - 부분 그래프는 원래의 그래프에 없는 정점이나 간선을 포함하지 않는다.



[그래프 G1]



[그래프 G1′]

그래프 이해

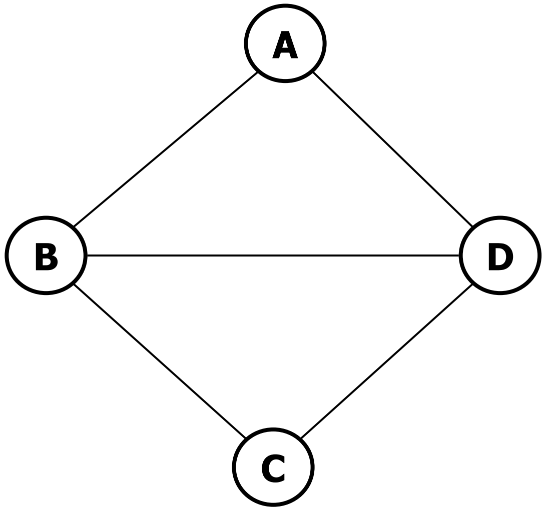
그래프 표현: 인접 행렬, 인접 리스트

그래프 표현: 인접 행렬 (1/3)

- **인접 행렬**(Adjacent Matrix)
  - 순차 자료구조를 이용하는 2차원 배열의 방법
    - **그래프의 두 정점을 연결한 간선의 유무를 행렬로 저장**
      - N 개의 정점을 가진 그래프: **N x N 정방 행렬**
      - **행렬의 행과 열:** 그래프의 정점
      - **행렬 값: 두 정점이 인접되어 있으면 1, 인접되어 있지 않으면 0**
  - **무향 그래프의 인접 행렬**
    - 행 i 의 합 = 열 i 의 합 = 정점 i 의 차수
  - **유향 그래프의 인접 행렬**
    - 행 i 의 합 = 정점 i 의 진출 차수
    - 열 i 의 합 = 정점 i 의 진입 차수

그래프 표현: 인접 행렬 (2/3)

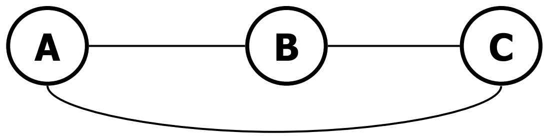
- **인접 행렬:** 2차원 배열



[그래프 G1]

	A	B	C	D
A	0	1	0	1
B	1	0	1	1
C	0	1	0	1
D	1	1	1	0

정점 B의 차수: 3 = 1+0+1+1

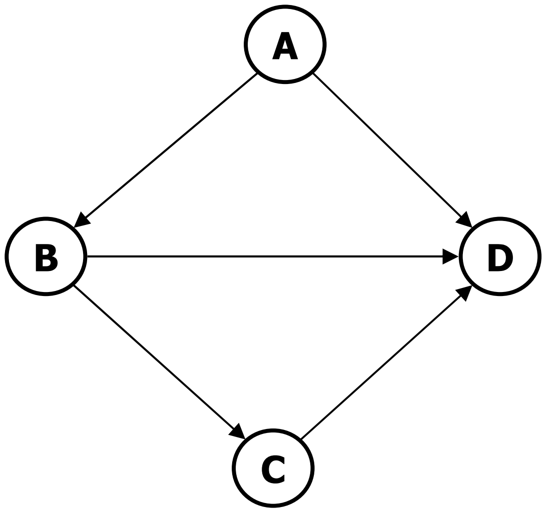


[그래프 G2]

	A	B	C
A	0	1	1
B	1	0	1
C	1	1	0

그래프 표현: 인접 행렬 (3/3)

- **인접 행렬:** 2차원 배열

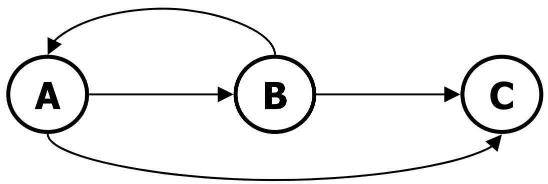


[그래프 G3]

	A	B	C	D
A	0	1	0	1
B	0	0	1	1
C	0	0	0	1
D	0	0	0	0

정점 B 의 진출 차수: 2 = 0+0+1+1

정점 B 의 진입 차수: 1 = 1+0+0+0



[그래프 G4]

	A	B	C
A	0	1	1
B	1	0	1
C	0	0	0

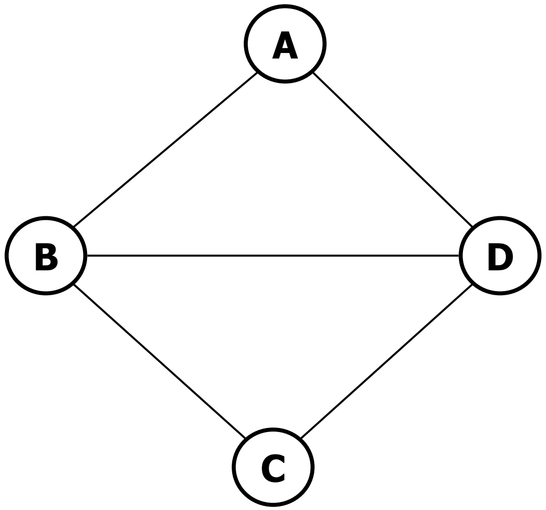
그래프 표현: 인접 리스트 (1/3)

- **인접 리스트**(Adjacent List)
  - 각 정점에 대한 인접 정점들을 연결하여 만든 단순 연결 리스트
  - **정점의 헤드 노드**
    - 정점에 대한 리스트의 시작을 표현
  - **인접 리스트의 각 노드**
    - 정점을 저장하는 필드와 다음 인접 정점을 연결하는 링크 필드로 구성
  - 각 정점의 차수만큼 노드를 연결
    - **리스트 내의 노드들은 인접 정점에 대해서 오름 차순으로 연결**

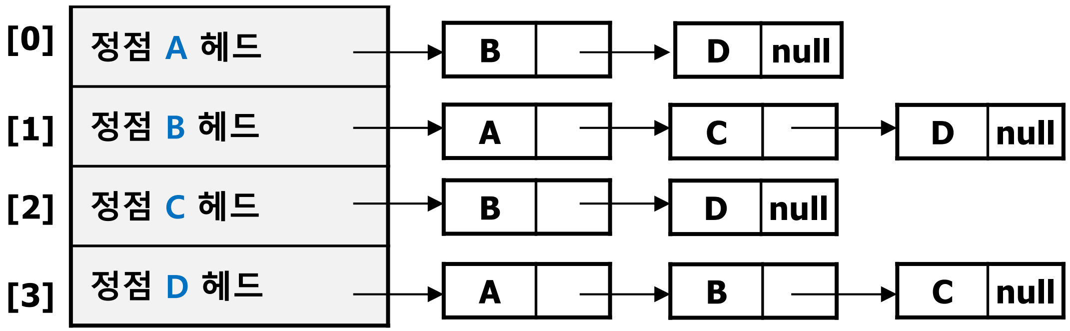
그래프 표현: 인접 리스트 (2/3)

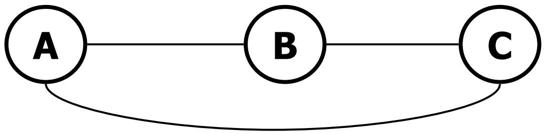
- **인접 리스트:** 무향 그래프

n 개의 정점(V)과 e 개의 간선(E)을 가진 무향 그래프
헤드 노드의 배열 크기: n
연결하는 노드의 총 수: 2e
각 정점의 헤드에 연결된 노드의 수: 정점의 차수

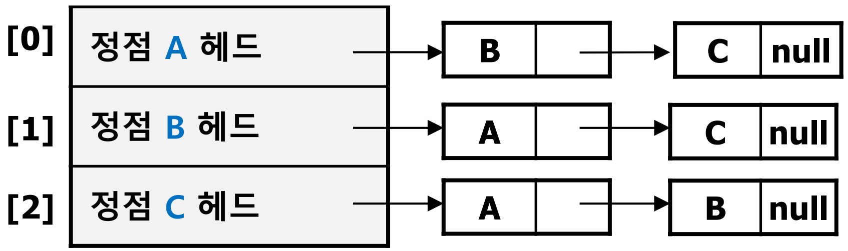


[그래프 G1]





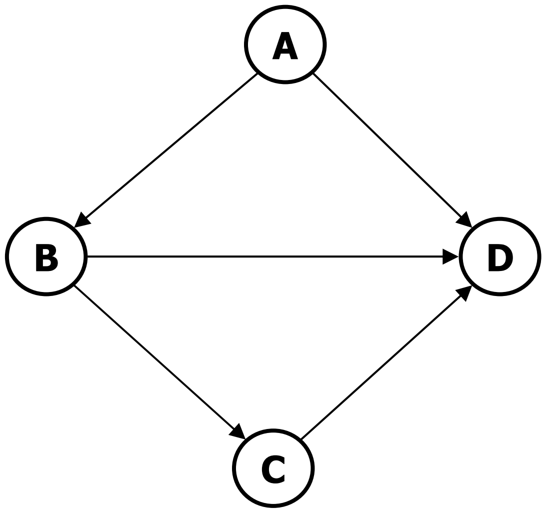
[그래프 G2]



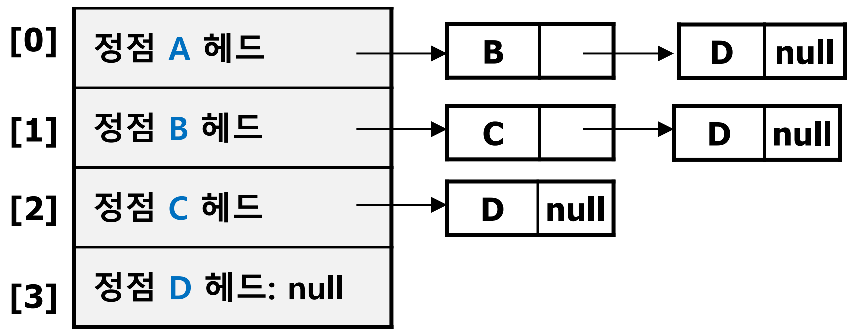
그래프 표현: 인접 리스트 (3/3)

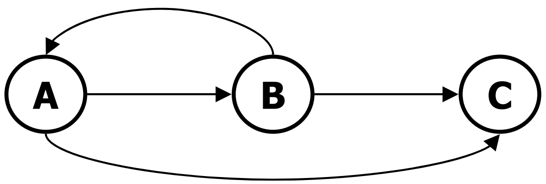
- **인접 리스트:** 유향 그래프

n 개의 정점(V)과 e 개의 간선(E)을 가진 유향 그래프
헤드 노드의 배열 크기: n
연결하는 노드의 총 수: e
각 정점의 헤드에 연결된 노드의 수: 정점의 진출 차수

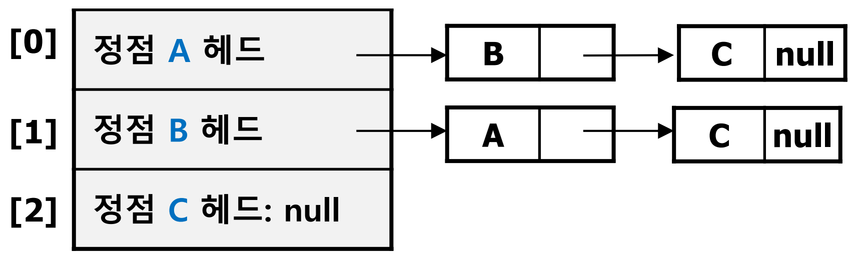


[그래프 G3]





[그래프 G4]



그래프의 이해

그래프 순회

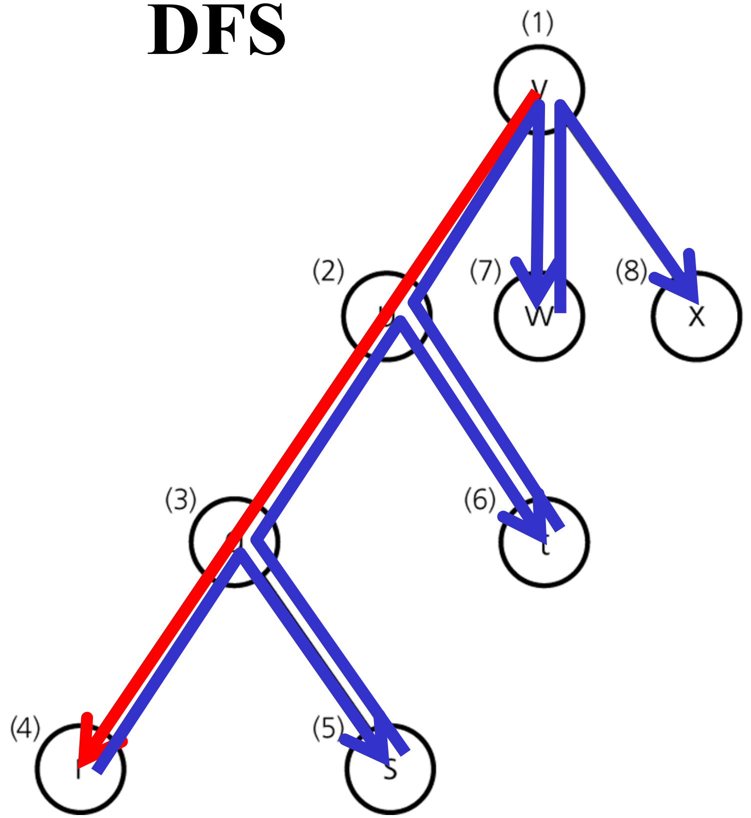
그래프 순회 (1/6)

- **그래프 순회(Graph Traversal)**
  - 그래프 탐색(Graph Search)
    - 하나의 정점에서 시작하여 그래프에 있는 모든 정점을 한번씩 방문한다.
  - 그래프 순회의 두 가지 방법
    - 깊이 우선 탐색(DFS, Depth First Search)
    - 너비 우선 탐색(BFS, Breadth First Search)

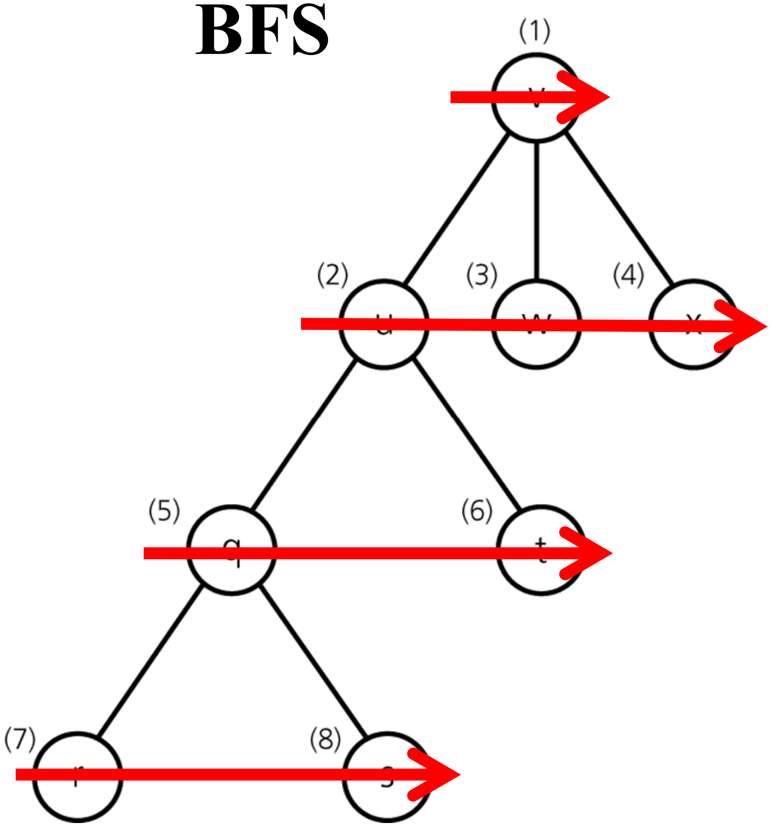
그래프 순회 (2/6)

- **이진 트리의 순회**
  - 깊이 우선 순회(DFS)와 너비 우선 순회(BFS)

DFS



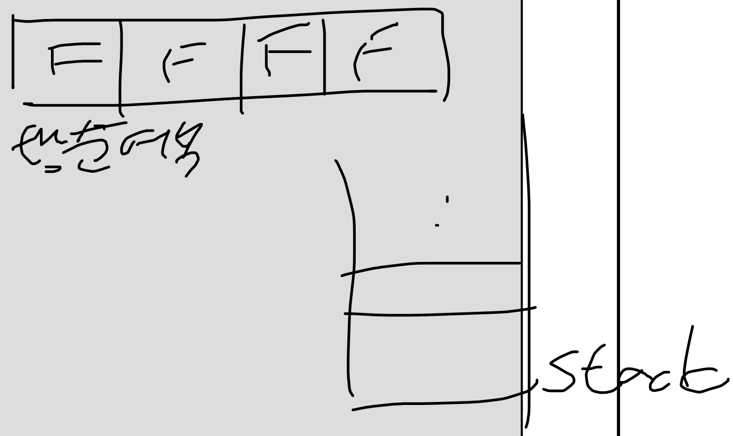
BFS



그래프 순회 (3/6)

- **깊이 우선 탐색(DFS, Depth First Search)**

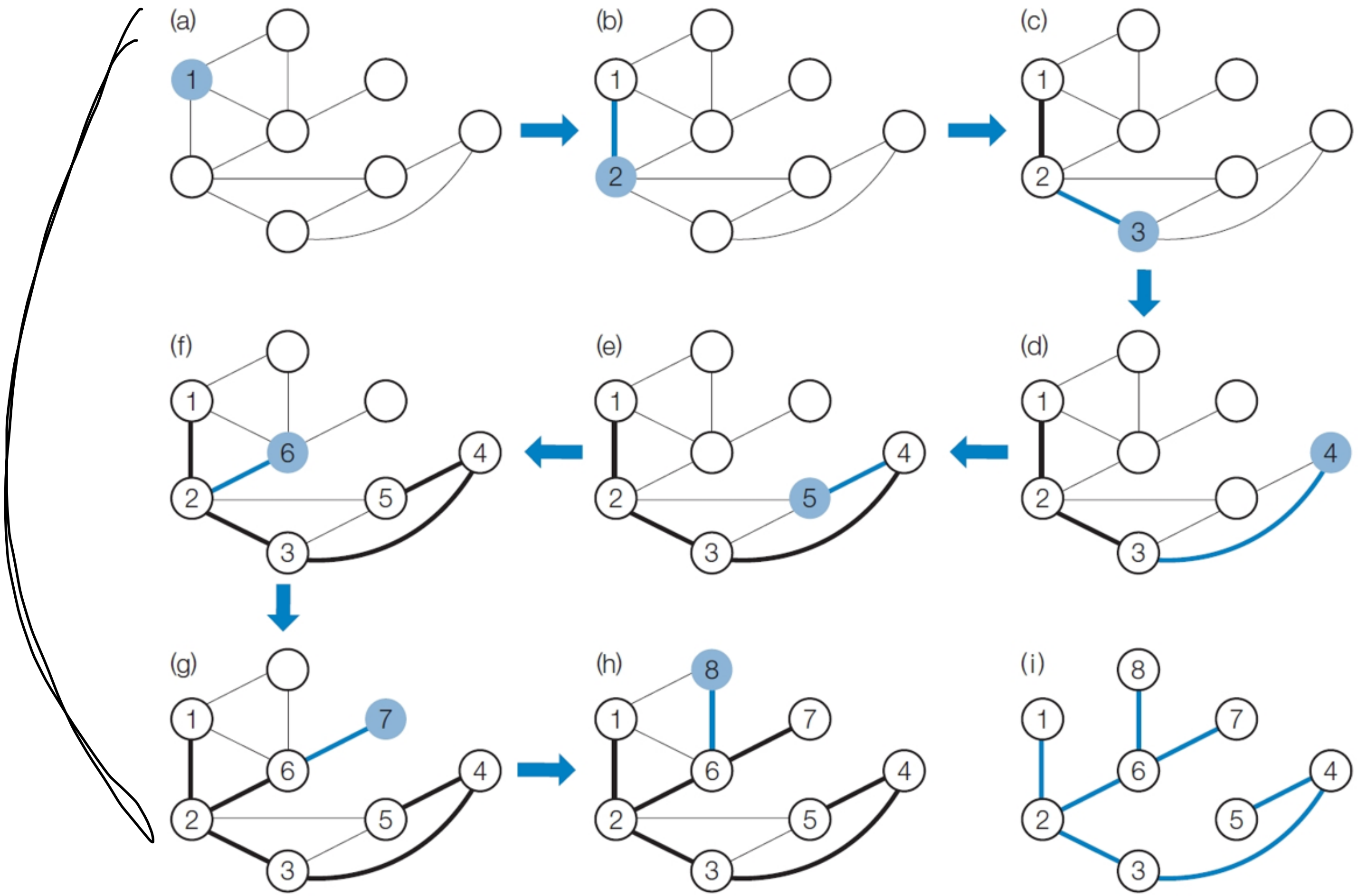
```
// 재귀적 용법
DFS(G):
    for each v ∈ V      visited[v] ← NO;
    for each v ∈ V
            if (visited[v] = NO) then aDFS(v);

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




```
// 비재귀적 용법
DFS(v):     ◄ 모든 정점은 초반에 UNVISITED로 마크 됨.
    stack.push(v)
    mark[v] ← VISITED
    while (!stack.isEmpty())
            if (no unvisited vertices are adjacent to the stack-top vertex)
                        stack.pop()                ◄ backtracking
            else
                        select an unvisited vertex w adjacent to the stack-top vertex
                        stack.push(w)          mark[w] ← VISITED
```

그래프 순회 (4/6)

- **깊이 우선 탐색:** 동작과정



그래프 순회 (5/6)

- **너비 우선 탐색(BFS, Breadth First Search)**

```
// 비재귀적 용법
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);
```

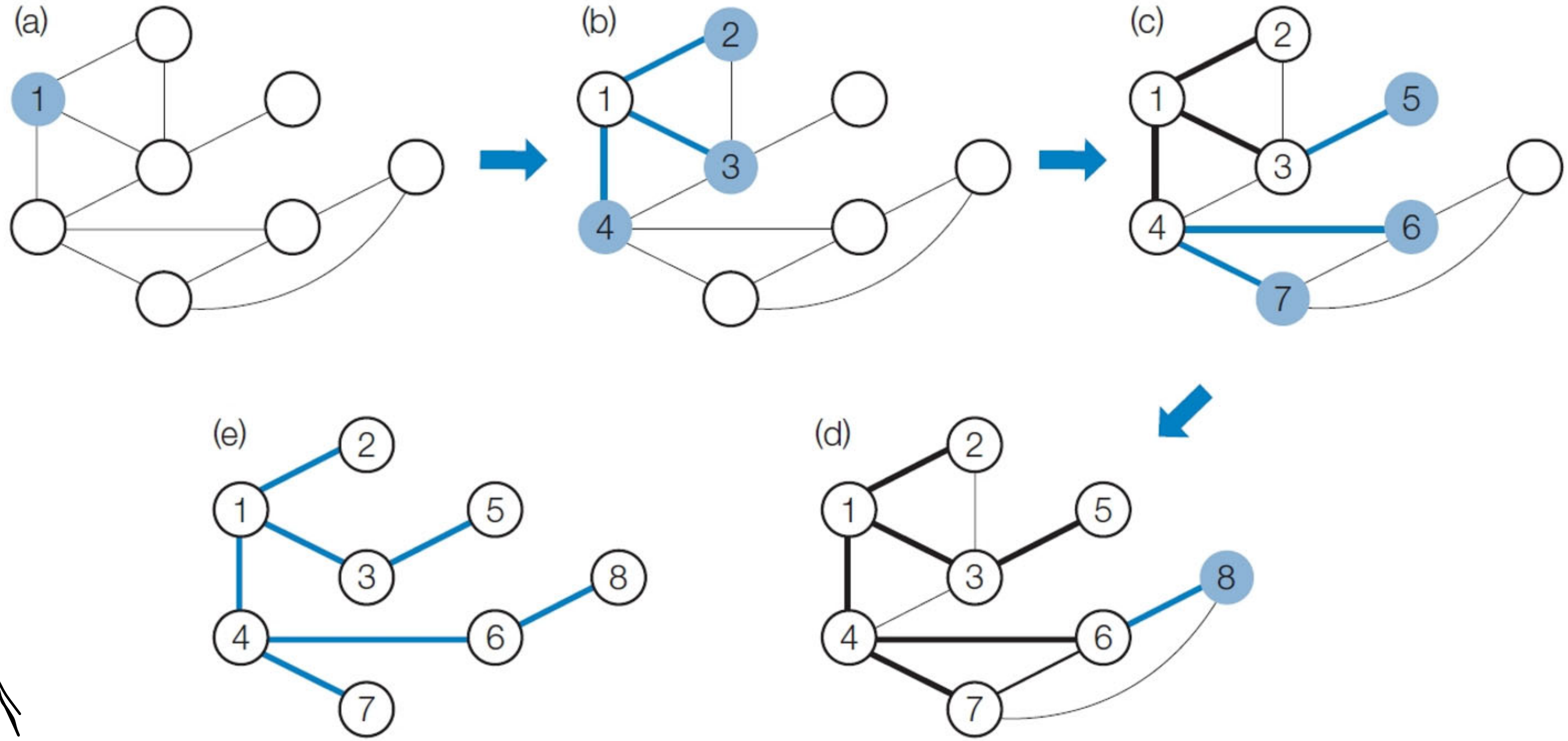
F F F F

방문여부

queue

그래프 순회 (6/6)

- **너비 우선 탐색:** 동작 과정



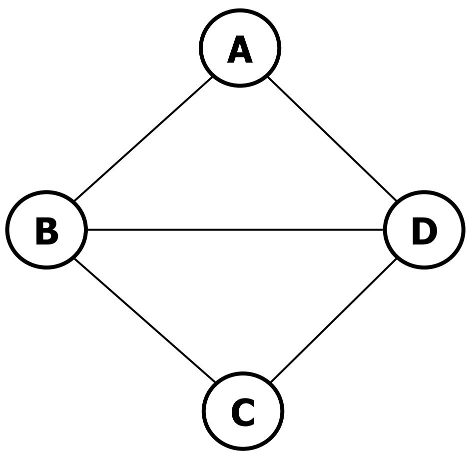
[이미지 출처: "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 문병로, 한빛아카데미, 2022.]

최소 신장 트리

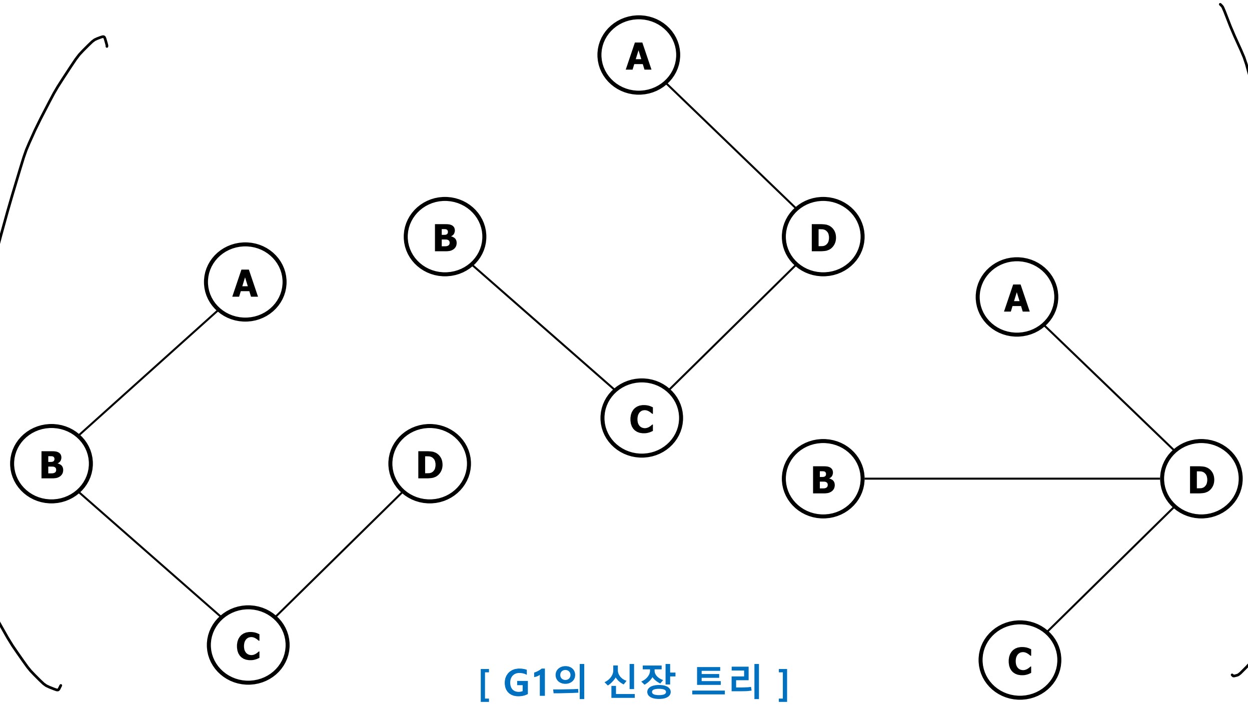
- 그래프의 이해
- **최소 신장 트리**
  - 신장 트리
  - Prim 알고리즘
  - Kruskal 알고리즘
- 위상 정렬
- 최단 경로

신장 트리 (1/6)

- **신장 트리(Spanning Tree)**
  - N 개의 정점으로 이루어진 무향 그래프 G 에서 N 개의 모든 정점과 N-1 개의 간선으로 만들어진 트리
    - 그래프의 관점에서 트리는 사이클이 없는 단순 연결 그래프



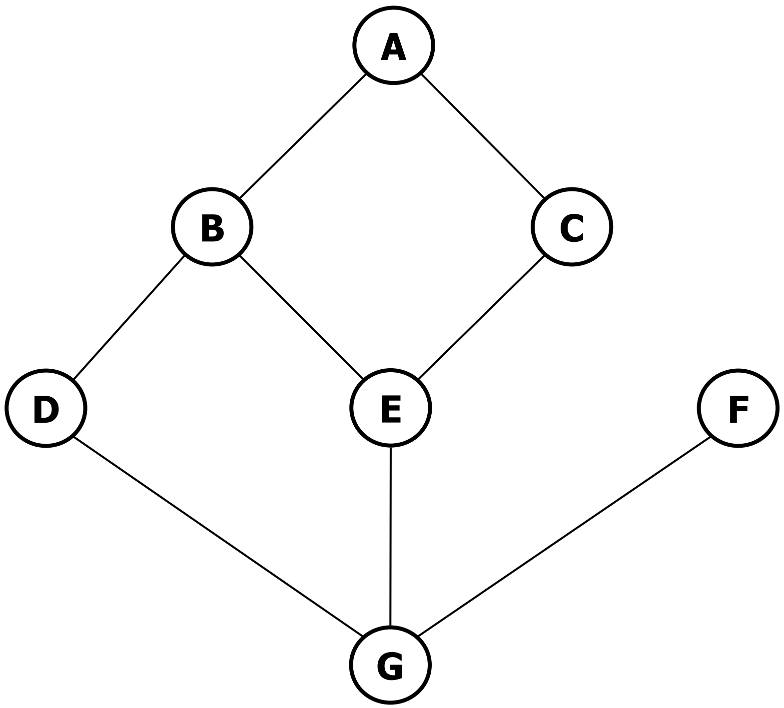
[그래프 G1]

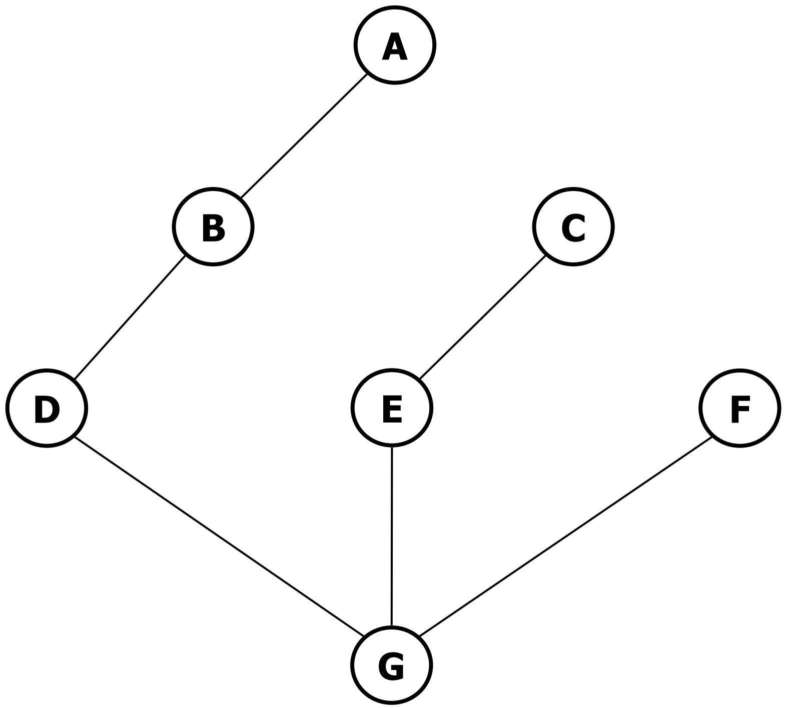


[G1의 신장 트리]

신장 트리 (2/6)

- **신장 트리:** 깊이 우선 신장 트리
  - 깊이 우선 신장 트리(Depth First Spanning Tree)
    - 깊이 우선 탐색을 이용하여 생성된 신장 트리





[그래프 G9 와 깊이 우선 신장 트리]

신장 트리 (3/6)

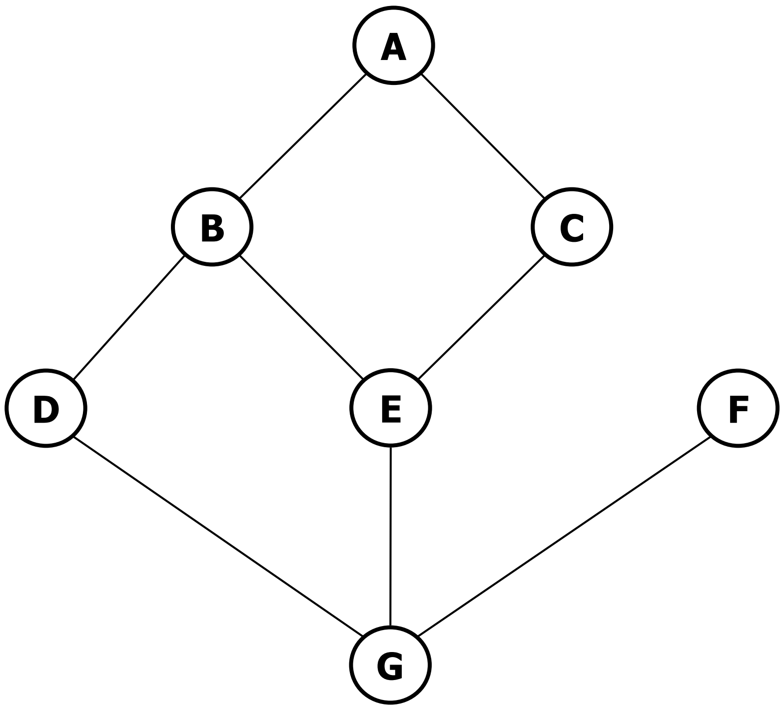
- **신장 트리:** 깊이 우선 신장 트리
  - **깊이 우선 신장 트리:** 알고리즘

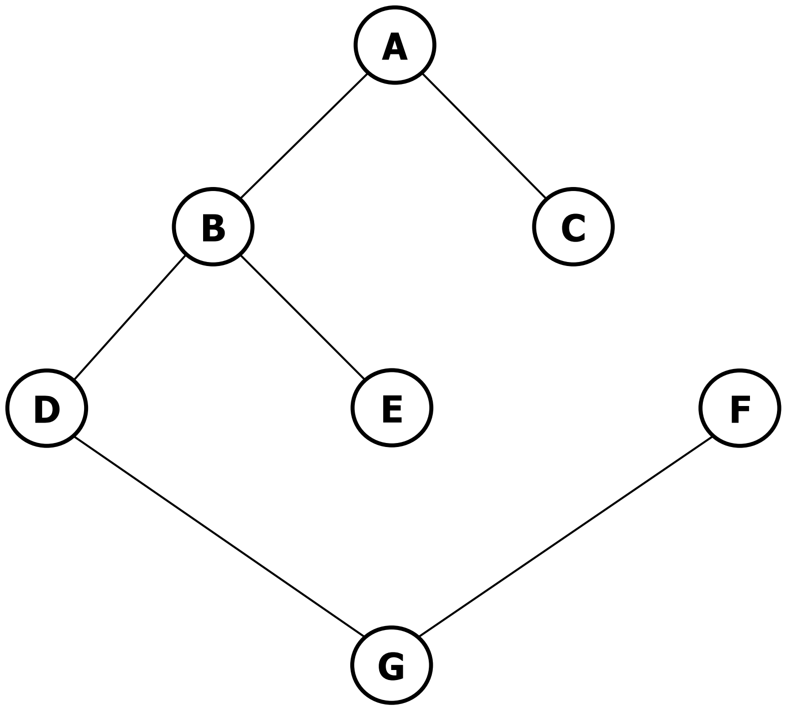
```
// 재귀적 용법
DFSTree(v):
        mark[v] ← VISITED
        for (each unvisited vertex u adjacent to v )
                T = T∪{(u -v)}
                DFSTree(u)
```

- **만들어진 T 는 DFS 신장 트리라 한다.**
- 보통은 간선을 **{u, v}** 또는 **(u, v)**로 표현한다.
- 여기서는 좀 더 직관적인 **(u-v), (u→v)** 로 표현하기로 한다.

신장 트리 (4/6)

- **신장 트리:** 너비 우선 신장 트리
  - 너비 우선 신장 트리(Breadth First Spanning Tree)
    - 너비 우선 탐색을 이용하여 생성된 신장 트리





[그래프 G9와 너비 우선 신장 트리]

신장 트리 (5/6)

- **신장 트리:** 깊이 우선 신장 트리
  - **깊이 우선 신장 트리:** 알고리즘

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

- **만들어진 T 는 DFS 신장 트리라 한다.**

- 보통은 간선을 **{u, v}** 또는 **(u, v)**로 표현한다.
- 여기서는 좀 더 직관적인 **(u-v), (u→v)** 로 표현하기로 한다.

신장 트리 (6/6)

- **최소 신장 트리(Minimum Spanning Tree)**
  - 무향 가중치 그래프에서 신장 트리를 구성하는 간선들의 가중치 합이 최소인 신장 트리
    - **가중치 그래프의 간선에 주어진 가중치**
      - 비용이나 거리, 시간을 의미하는 값
  - 최소 신장 트리 생성 알고리즘
    - **Prim 알고리즘**
    - **Kruskal 알고리즘**

최소 신장 트리

Prim 알고리즘

Prim 알고리즘 (1/4)

● Prim 알고리즘

- Prim 알고리즘은 간선을 정렬하지 않고, 하나의 정점에서 시작하여 트리를 확장해 나가는 방법
  - 프림 알고리즘은 그리디 알고리즘의 예이다.
    - **그리디 알고리즘(Greedy Algorithm):** 우선 눈앞의 이익을 최대화하는 선택을 계속하는 알고리즘을 총칭한다.

```
// 직관적 표현
Prim(v):
        Mark v as visited
        while (there are unvisited vertices)
                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)}
```

Prim 알고리즘 (2/4)

● Prim 알고리즘: 알고리즘

```
// Prim 알고리즘
Prim(G, r):
▷ G = (V, E): 주어진 그래프
▷ r : 시작 정점
        S ← Φ ;                    ▷ S : 정점 집합
        for each u ∈ V
                du ← ∞ ;
        dr ← 0 ;
        while (S ≠ V) {            ▷ n 회 순환 된다
                u ← extractMin(V-S, d) ;
                S ← S ∪{u};
                for each v∈L(u)     ▷ L(u) : u로부터 연결된 정점들의 집합
                        if (v∈V-S and wuv< dv) then dv ←  wuv ;
        }
extractMin(Q, d):
        집합 Q에서 d값이 가장 작은 정점 u를 리턴한다 ;
```

알고리즘 13-3 프림 알고리즘

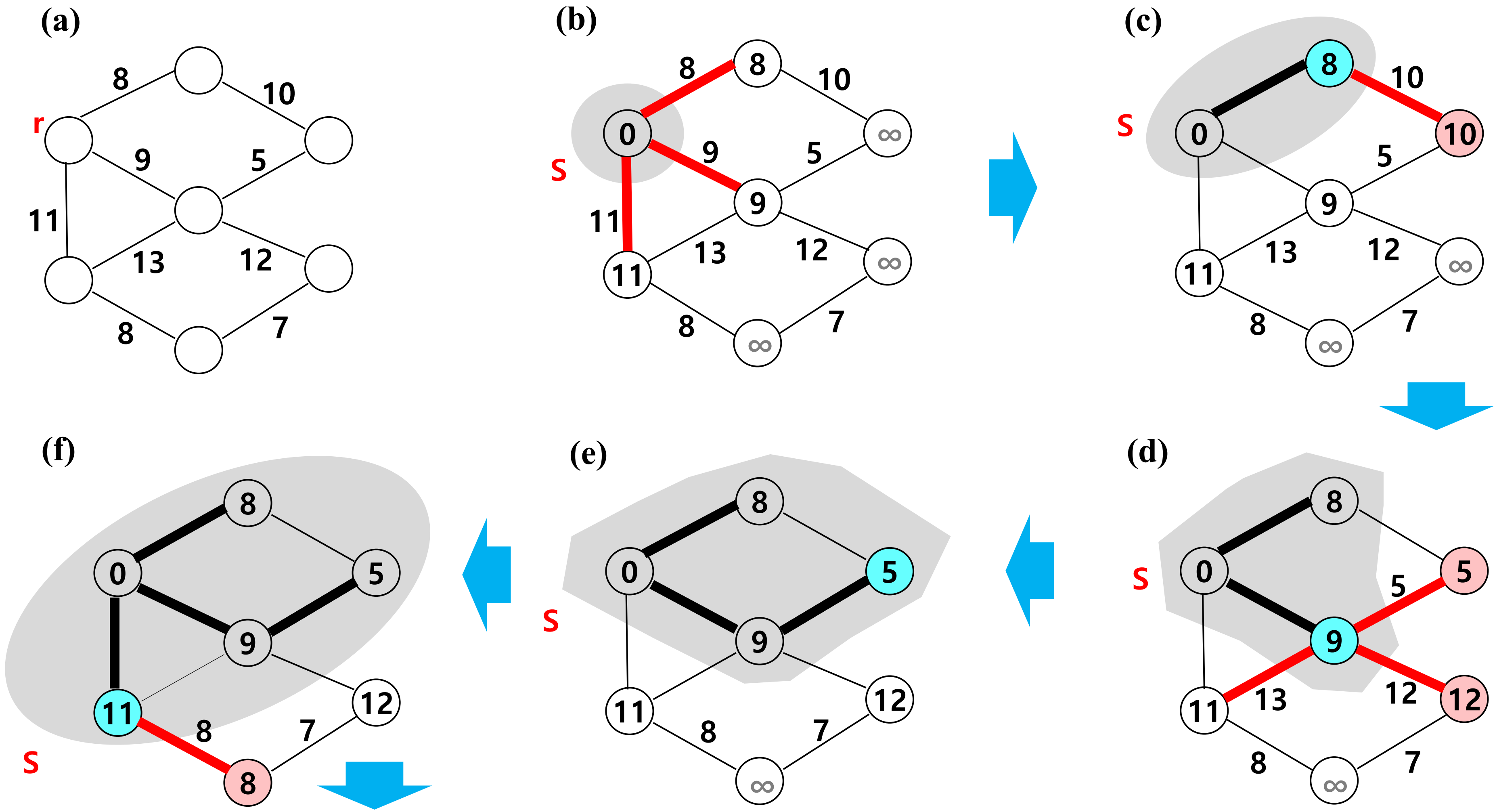
```
Prim(G, r):
◀ G = (V, E): 주어진 그래프,  r: 시작 정점
      S ← {r}  ◀ S: 정점 집합
      r.cost ← 0
    ❶ for each u ∈ V-{r}
        u.cost ← w_ru
    ❷ while (S≠V)               ◀ n-1회 순환한다.
      ❸ u ← deleteMin(V-S)
        S ← S∪{u}
      ❹ for each v ∈ u.adj      ◀ u.adj: 정점 u에 인접한 정점 집합
        ❺ if (v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

: 방금 S 에 포함된 정점

: 방금 이완이 일어난 정점

Prim 알고리즘 (3/4)

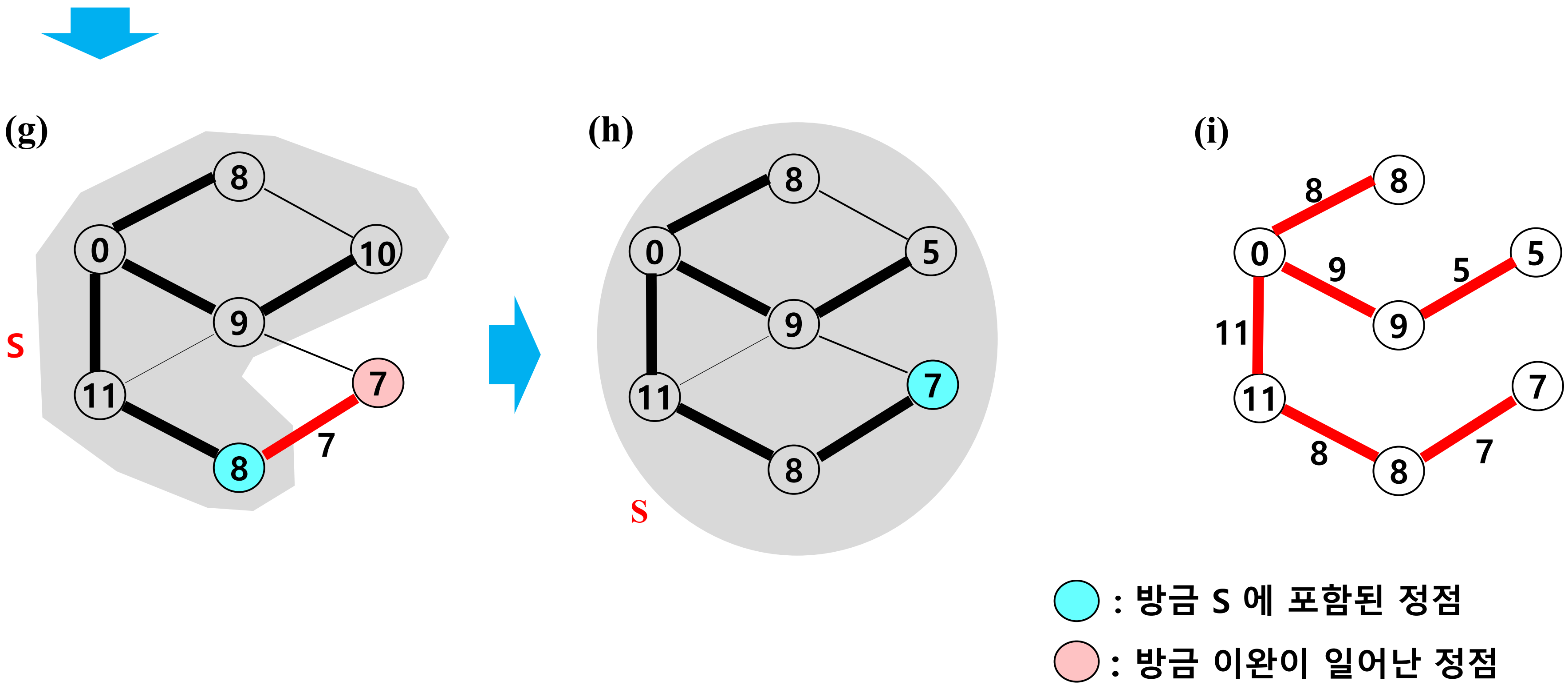
- **Prim 알고리즘:** 동작 과정 #1



[이미지 출처: 문병로, "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 한빛아카데미, 2022.]

Prim 알고리즘 (4/4)

- **Prim 알고리즘:** 동작 과정 #2



최소 신장 트리

Kruskal 알고리즘

Kruscal 알고리즘 (

알고리즘 13-4 크루스칼 알고리즘

```
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 알고리즘

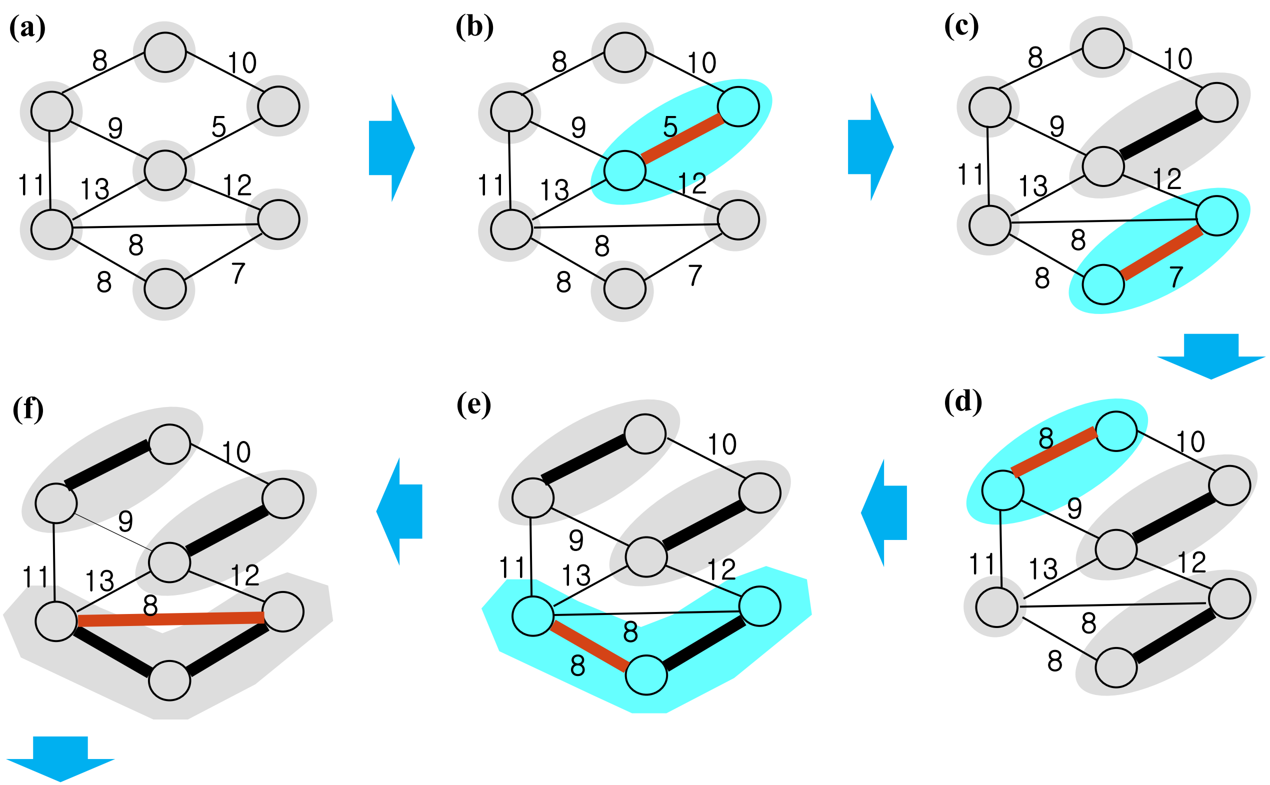
○ 크루스칼 알고리즘

- 크루스칼 알고리즘도 그리디 알고리즘의 예이다.
  - **그리디 알고리즘(Greedy Algorithm):** 우선 눈앞의 이익을 최대화하는 선택을 계속하는 알고리즘을 총칭한다.

```
// Krusacl 알고리즘
Kruskal (G, r):
        1. T ← Φ ;                ▷ T : 신장 트리
        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;
                정점 u와 정점 v가 서로 다른 집합에 속하면 {
                        두 집합을 하나로 합친다;
                        T ← T∪{(u, v)};
                }
        }
```

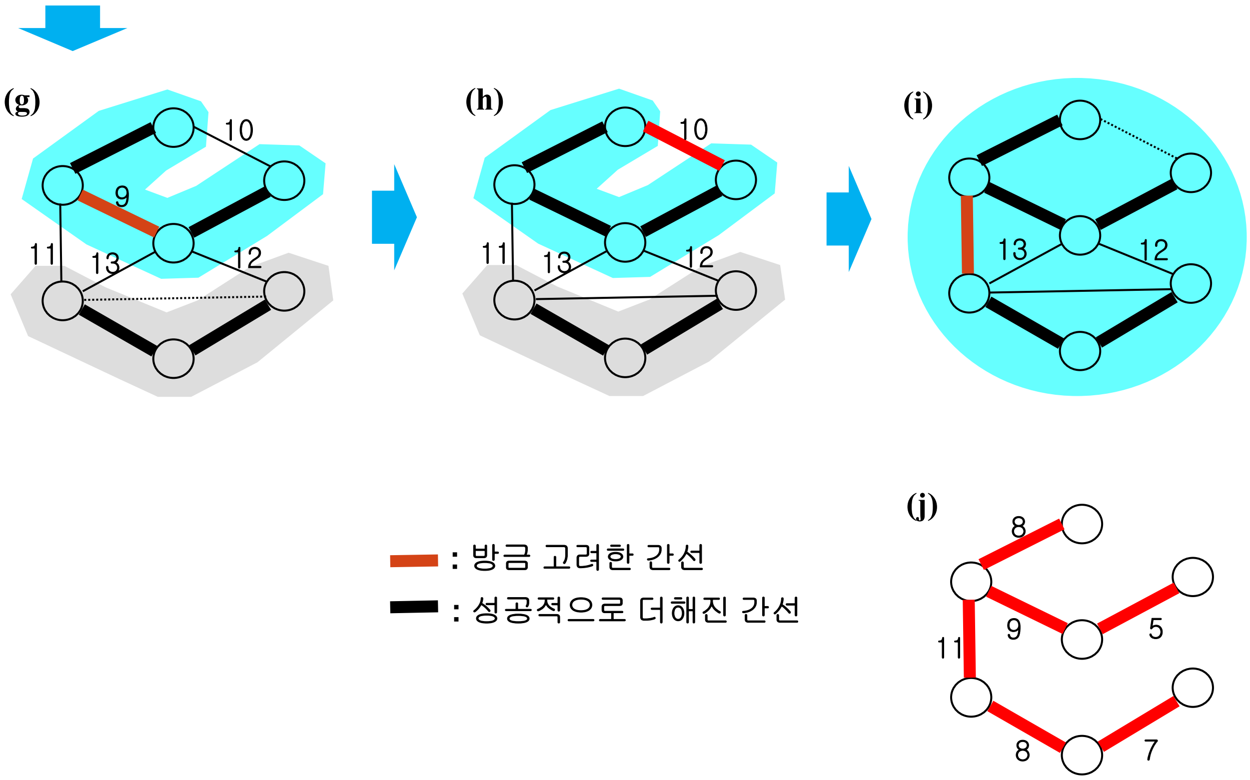
Kruscal 알고리즘 (2/3)

- **Kruscal: 동작 과정 #1**



Kruscal 알고리즘 (3/3)

- **Kruscal: 동작 과정 #2**



위상 정렬

- 그래프의 이해
- 최소 신장 트리
- **위상 정렬**
  - 위상 정렬
- 최단 경로

위상 정렬 (1/6)

- **위상 정렬(Topological sorting)**
  - **조건: 싸이클이 없는 방향 그래프**
  - **위상 순서**
    - 간선 **(x→y)** 가 존재하면 정점 **x** 는 정점 **y** 에 앞선다.
    - 대개 한 방향 그래프에는 서로 다른 위상 순서가 여럿 존재한다.
  - **위상 정렬Topological Sorting**
    - 주어진 방향 그래프 G의 위상 순서 중 하나를 찾는다

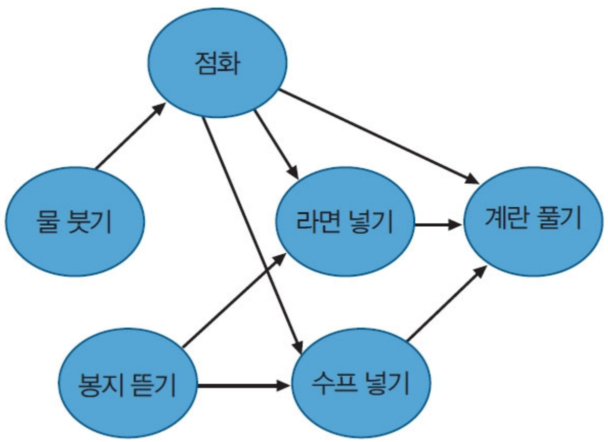


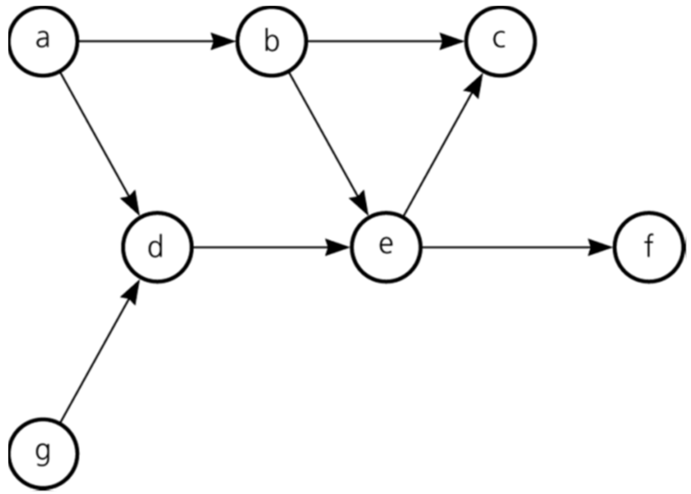
그림 13-17 라면 끓이기 작업의 선후 관계

순서 1	순서 2	순서 3
① 물 붓기	① 봉지 뜯기	① 봉지 뜯기
② 점화	② 물 붓기	② 물 붓기
③ 봉지 뜯기	③ 점화	③ 점화
④ 라면 넣기	④ 라면 넣기	④ 수프 넣기
⑤ 수프 넣기	⑤ 수프 넣기	⑤ 라면 넣기
⑥ 계란 풀기	⑥ 계란 풀기	⑥ 계란 풀기

그림 13-18 가능한 라면 끓이기 순서의 예

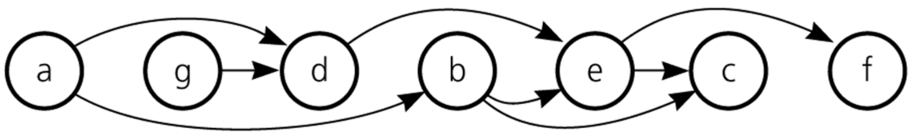
위상 정렬 (2/6)

- **위상 정렬**
  - 위상 정렬

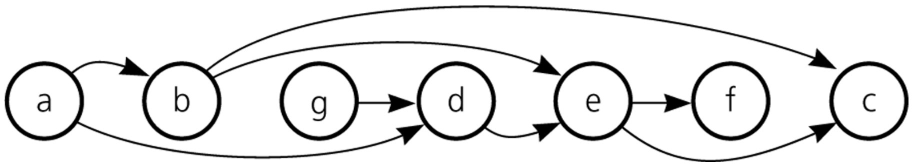


- 위 그래프에 대한 위상 정렬의 예: 2개(a, b)

(a)



(b)



위상 정렬 (3/6)

- **위상 정렬: 알고리즘**

알고리즘 13-5 위상 정렬 알고리즘

```
topologicalSort(G):
    for i ← 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 정렬 알고리즘
topologicalSort1(G, v):
    for ← 1 to n {
        진입 간선이 없는 정점 u를 선택한다;
         A[i] ← u;
        정점 u와, u의 진출 간선을 모두 제거한다;
    }
    ▷ 이 시점에 배열 A[1...n]에는 정점들이 위상 정렬되어 있다.
```

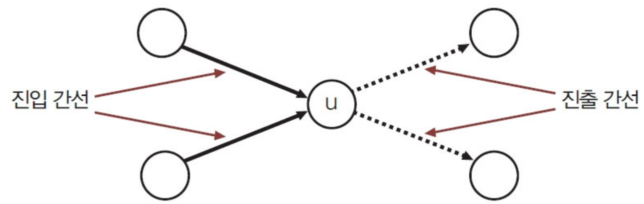


그림 13-19 정점 u의 진입 간선과 진출 간선

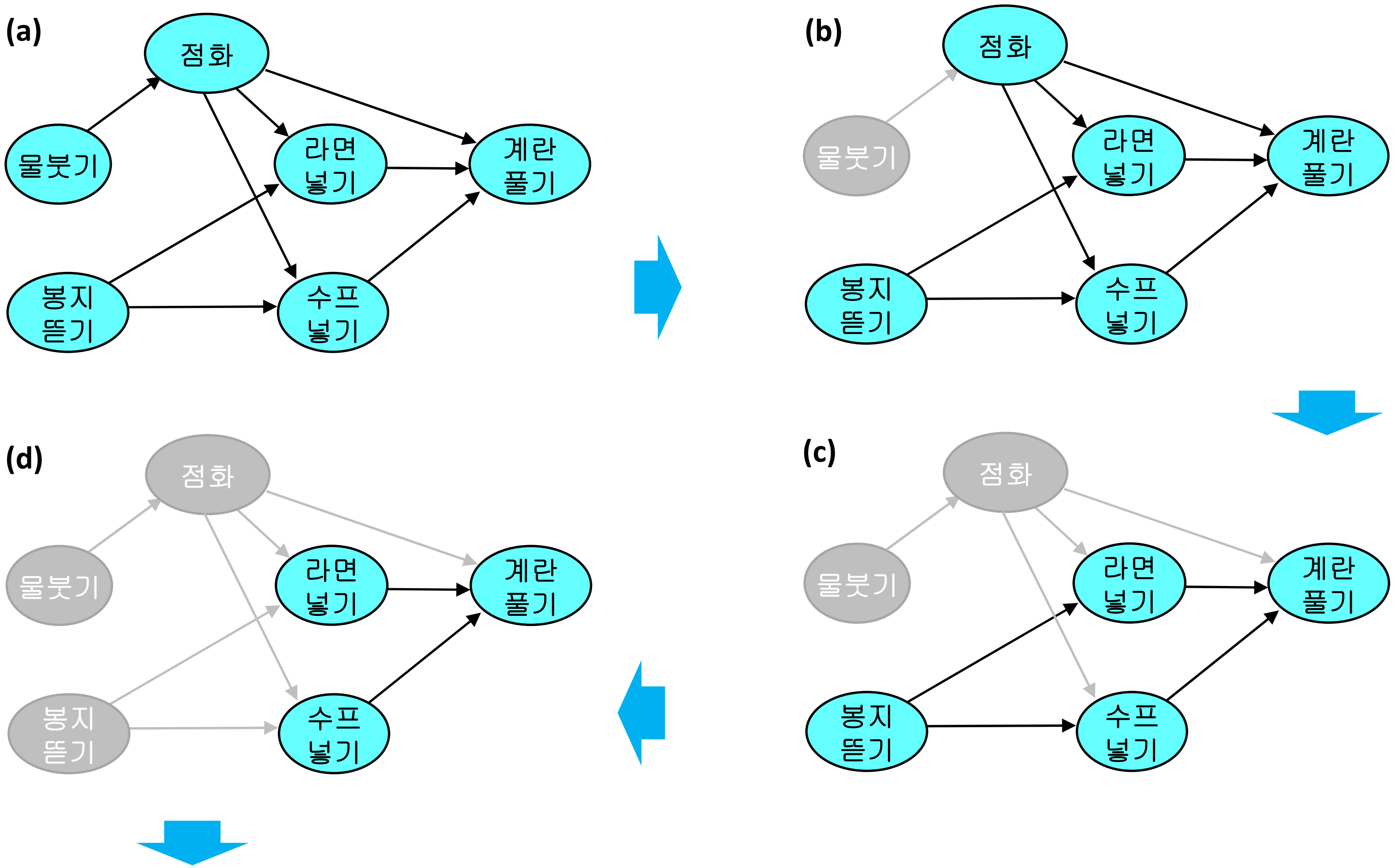
위상 정렬 (4/6)

- **위상 정렬:** 알고리즘

```
// 위상 정렬 알고리즘
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;
```

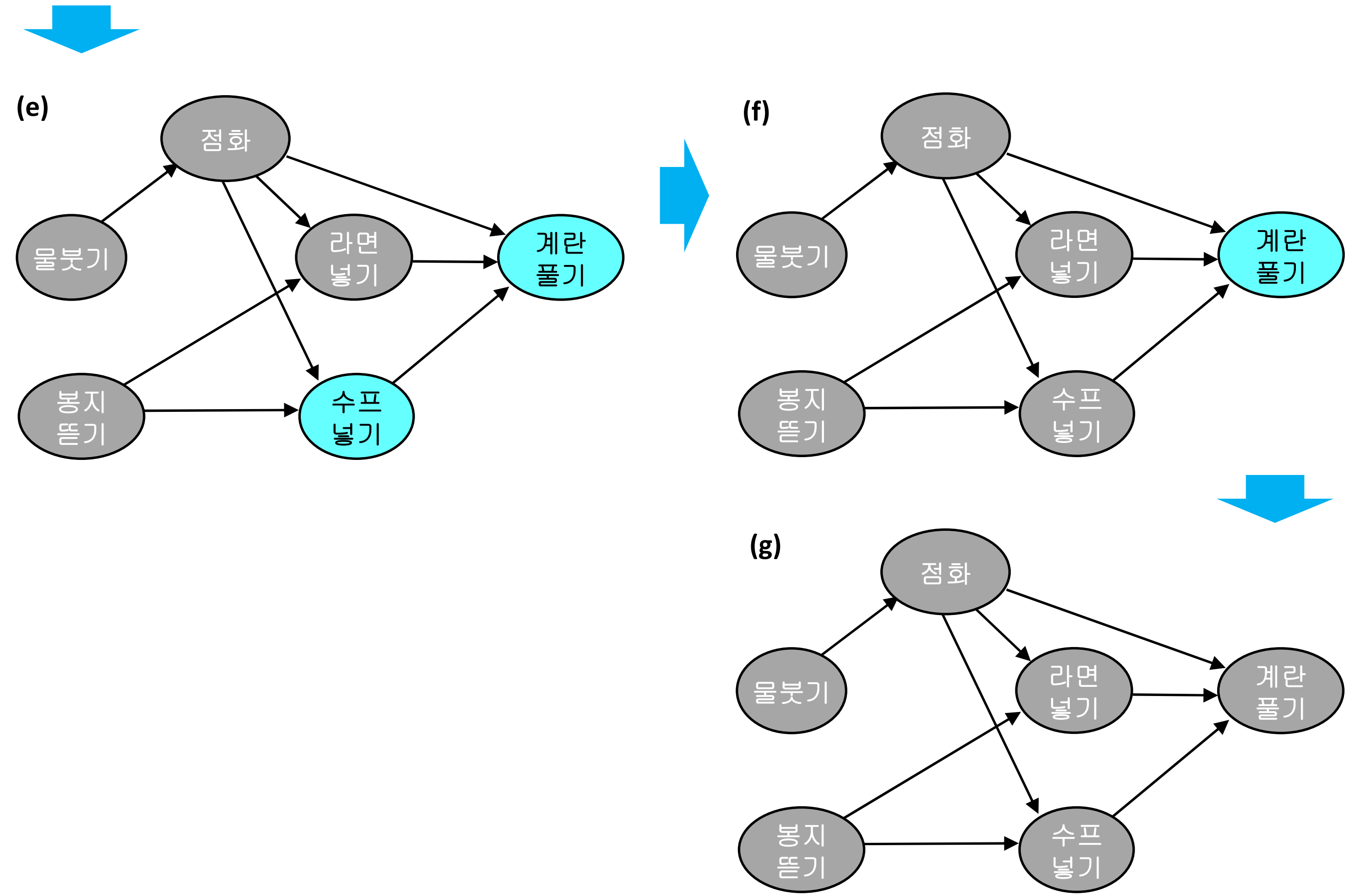
위상 정렬 (5/6)

- **위상 정렬: 동작 과정 #1**



위상 정렬 (6/6)

- **위상 정렬:** 동작 과정 #2



[이미지 출처: 문병로, “IT CookBook, 쉽게 배우는 자료구조 with 파이썬”, 한빛아카데미, 2022.]

최단 경로

- 그래프의 이해
- 최소 신장 트리
- 위상 정렬
- **최단 경로**
  - 단일 시작점 최단 경로
  - 모든 쌍 최단 경로

최단 경로

- **최단 경로(Shortest Paths)**
  - **조건:**
    - **간선 가중치가 있는 유향 그래프**
      - 무향 그래프는 각 간선에 대해 양쪽으로 유향 간선이 있는 유향 그래프로 생각할 수 있다.
      - 즉, 무향 간선 (u, v)는 유향 간선 (u, v)와 (v, u)를 의미한다고 가정하면 된다.
  - **두 정점 사이의 최단경로**
    - **두 정점 사이의 경로들 중 간선의 가중치 합이 최소인 경로**
      - **간선 가중치의 합이 음인 싸이클이 있으면 문제가 정의되지 않는다.**
  - **단일 시작점 최단경로**
    - **단일 시작점으로부터 각 정점에 이르는 최단경로를 구한다.**
    - **다익스트라 알고리즘:** 음의 가중치를 허용하지 않는 최단경로
    - **벨만-포드 알고리즘:** 음의 가중치를 허용하는 최단경로
    - **DAG 최단 경로:** 싸이클이 없는 유향 그래프의 최단경로
  - **모든 쌍 최단경로**
    - **모든 정점 쌍 사이의 최단경로를 모두 구한다.**
    - **플로이드-워샬 알고리즘**

최소 신장 트리

단일 시작점 최단 경로:

Dijkstra 알고리즘

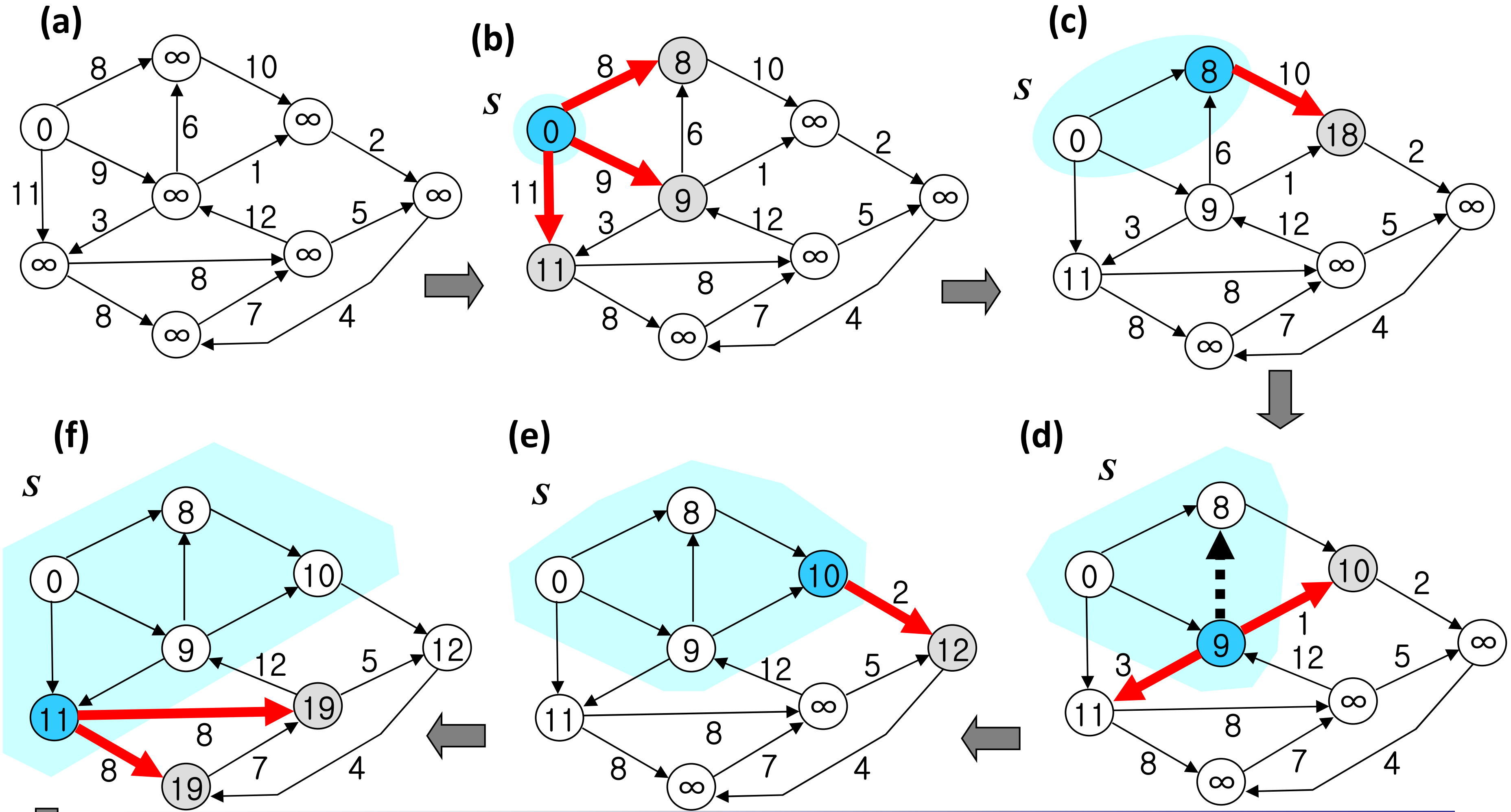
Dijkstra 알고리즘 (1/5)

- **Dijkstra 알고리즘:** 알고리즘

```
// Dijkstra 알고리즘
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 순환 된다.
                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;
        }

extractMin(Q, d[]):
    집합 Q에서 d값이 가장 작은 정점 u를 반환 한다 ;
```

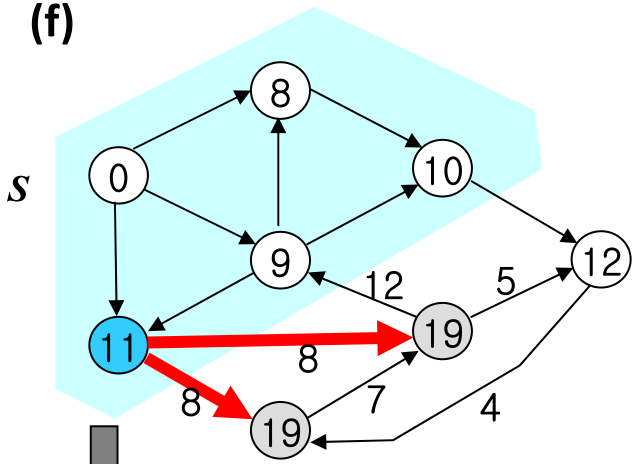
Dijkstra 알고리즘 (2/5)

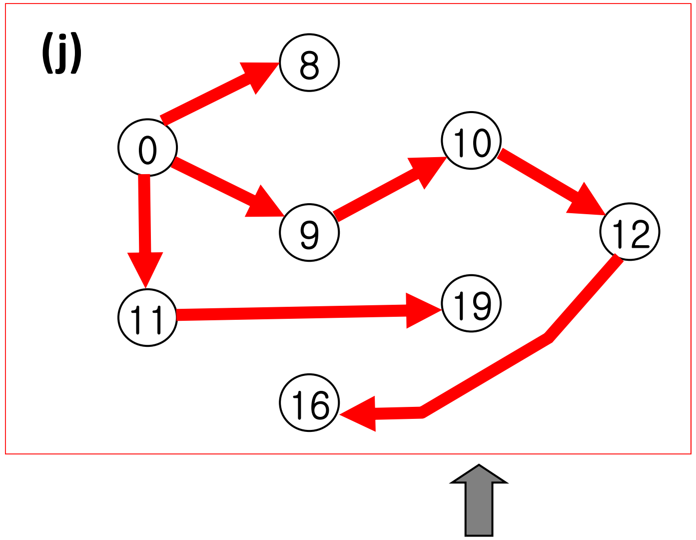
- **Dijkstra 알고리즘:** 동작 과정 #1

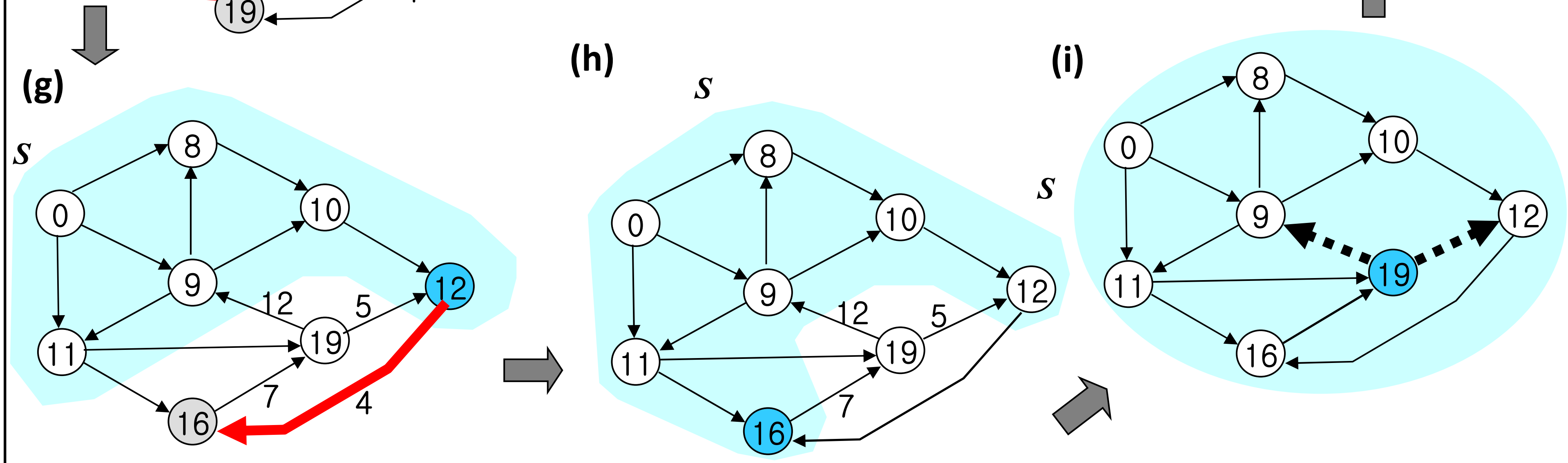


Dijkstra 알고리즘 (3/5)

- **Dijkstra 알고리즘:** 동작 과정 #2

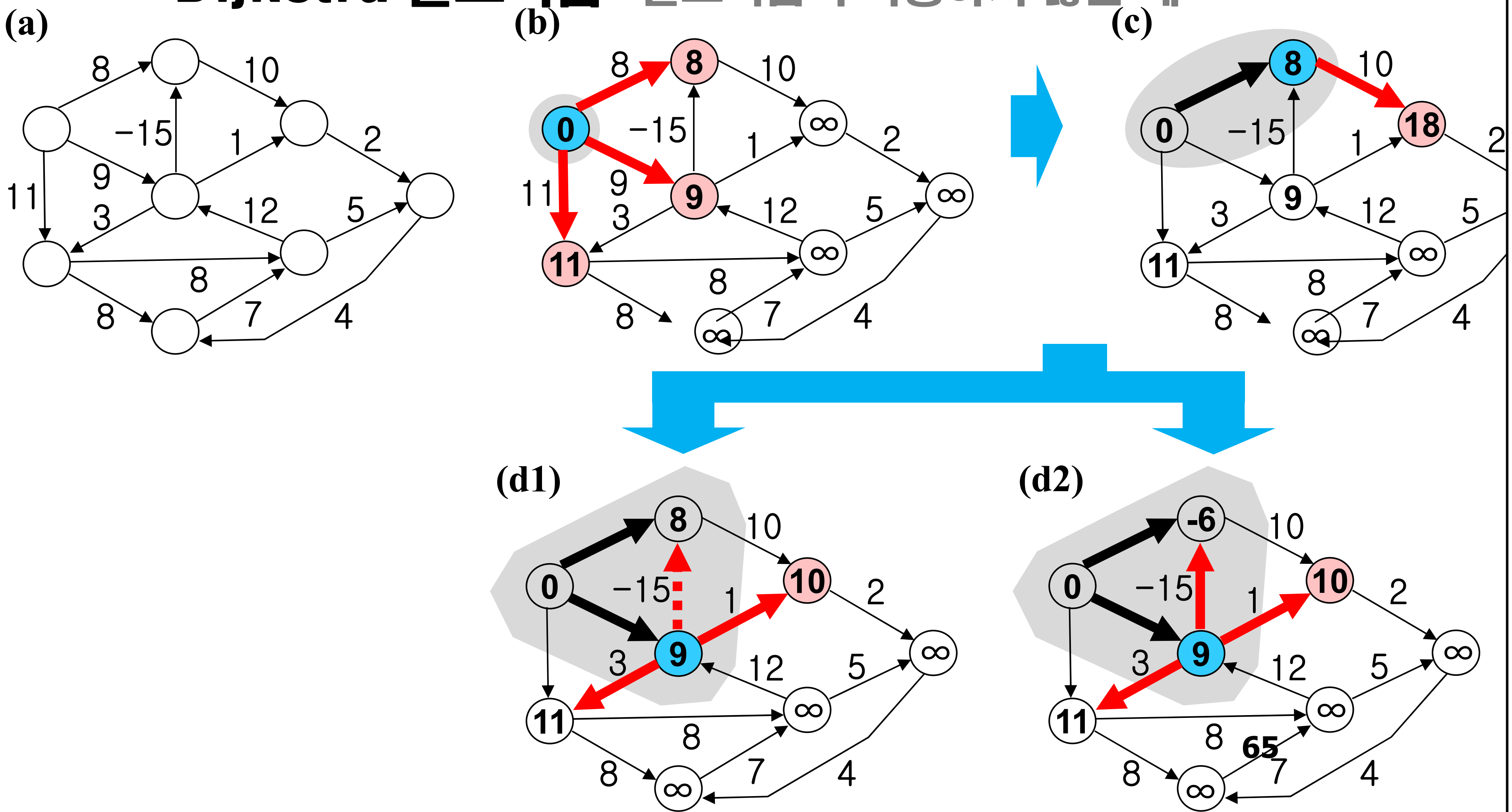






Dijkstra 알고리즘 (4/5)

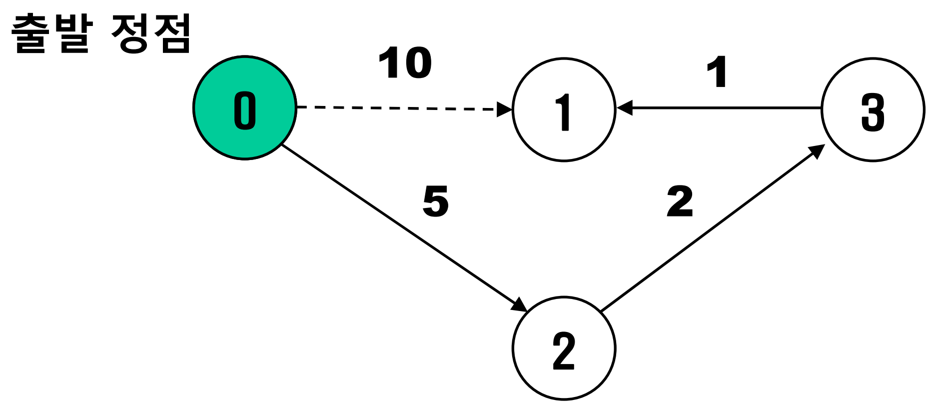
- **Dijkstra 알고리즘:** 알고리즘이 작동하지 않는 예



[이미지 출처: 문병로, "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 한빛아카데미, 2022.]

Dijkstra 알고리즘 (5/5)

- **Dijkstra 알고리즘:** 이완의 예



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

[이미지 출처: 문병로, "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 한빛아카데미, 2022.]

최소 신장 트리

단일 시작점 최단 경로:

Bellman-Ford 알고리즘

Bellman-Ford 알고리즘 (1/6)

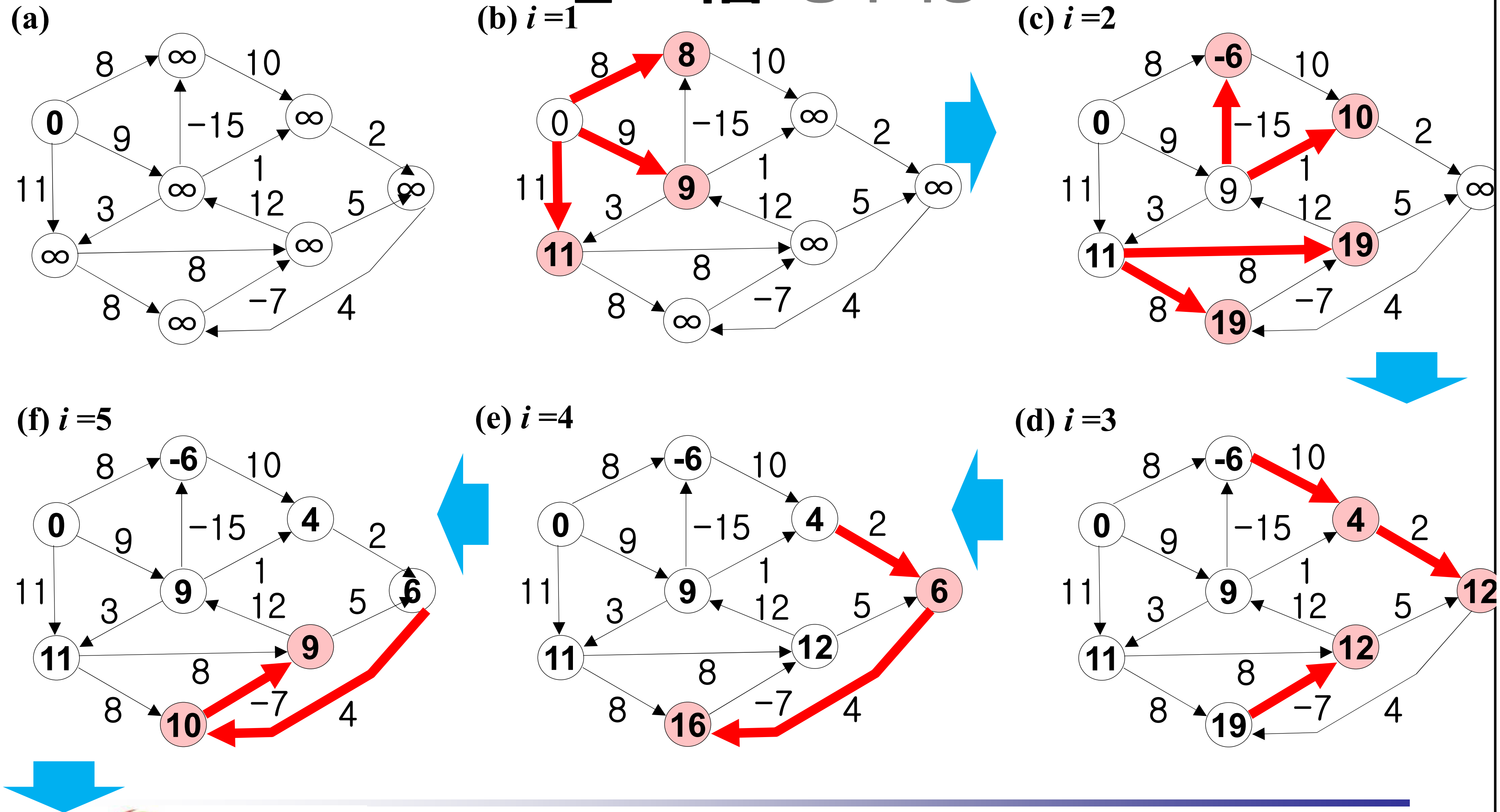
- **Bellman-Ford 알고리즘:** 알고리즘

```
// Bellman-Ford 알고리즘
BellmanFord(G, r):
        for each u∈V
                du← ∞;
        d[r] ← 0;
        for i ← 1 to |V|-1
                for each (u, v) ∈E
                        if (d[u] + w[u, v] < d[v]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]v )  output "해없음";
```

이완(relaxation)

Bellman-Ford 알고리즘 (2/6)

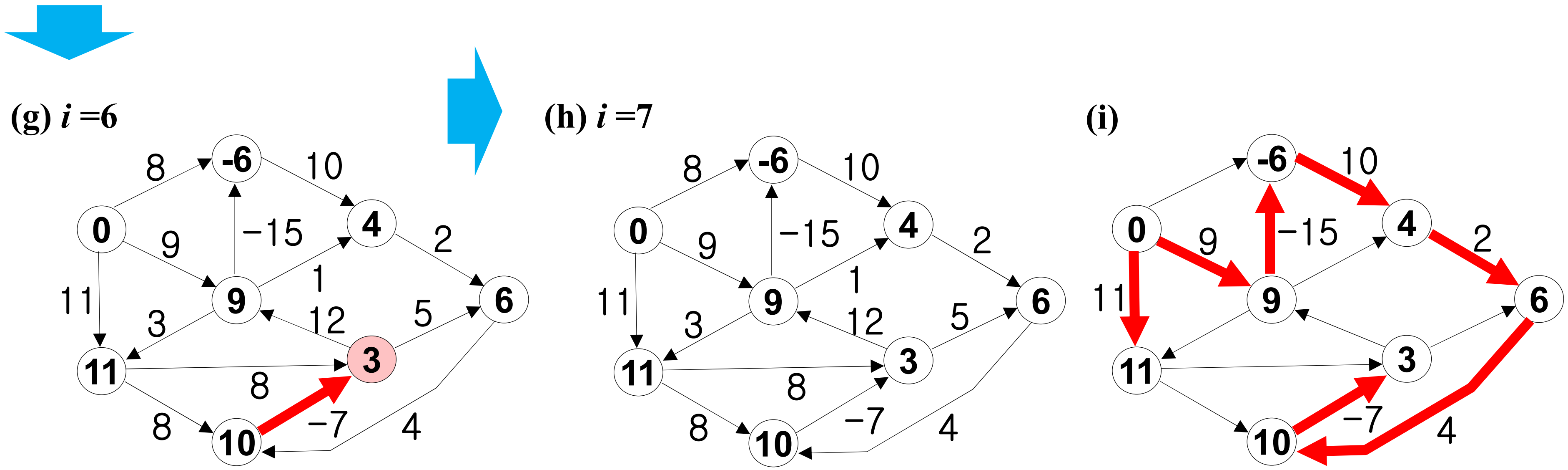
- **Bellman-Ford 알고리즘:** 동작 과정 #1



[이미지 출처: 문병로, "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 한빛아카데미, 2022.]

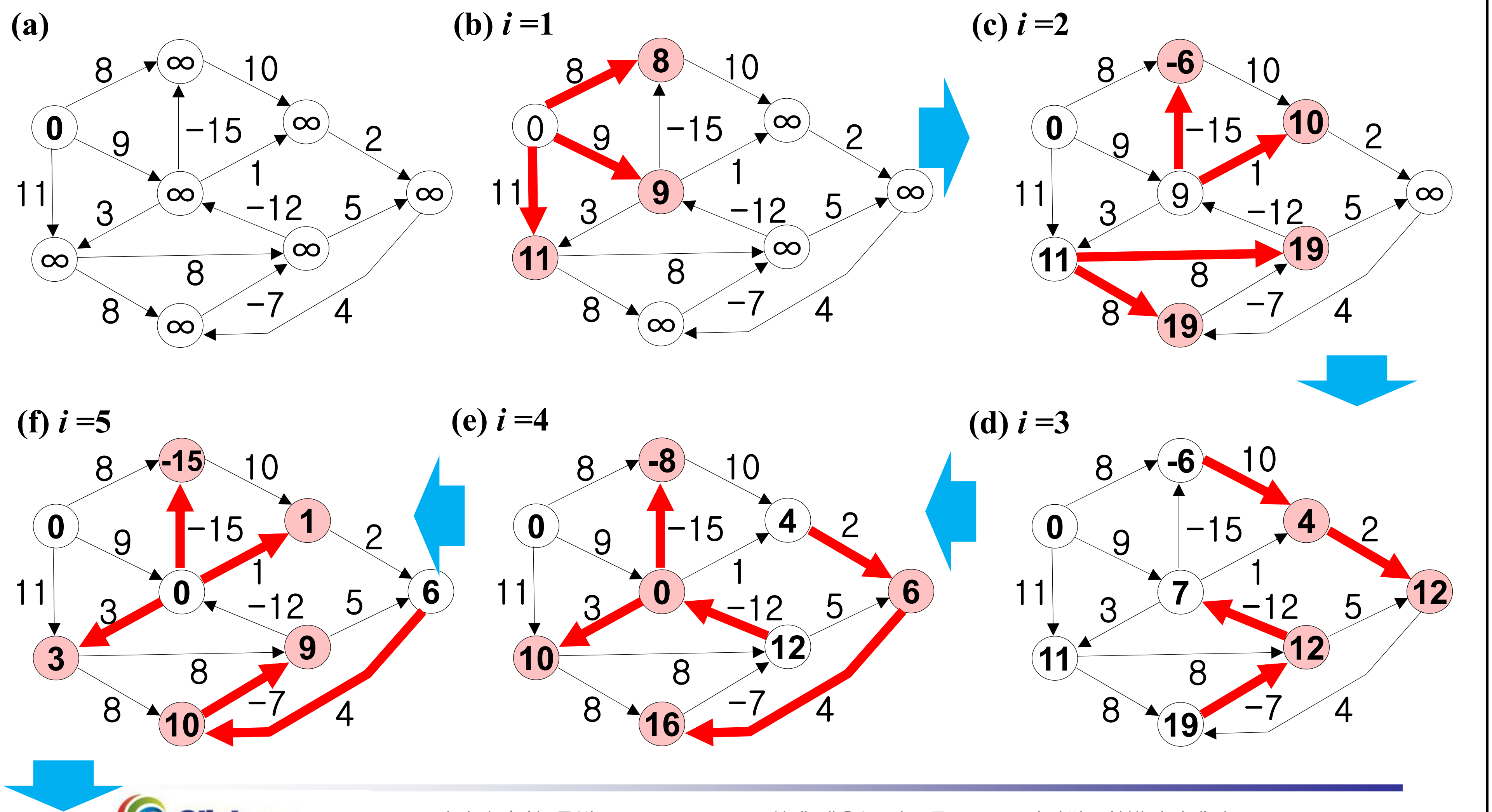
Bellman-Ford 알고리즘 (3/6)

- **Bellman-Ford 알고리즘:** 동작 과정 #2



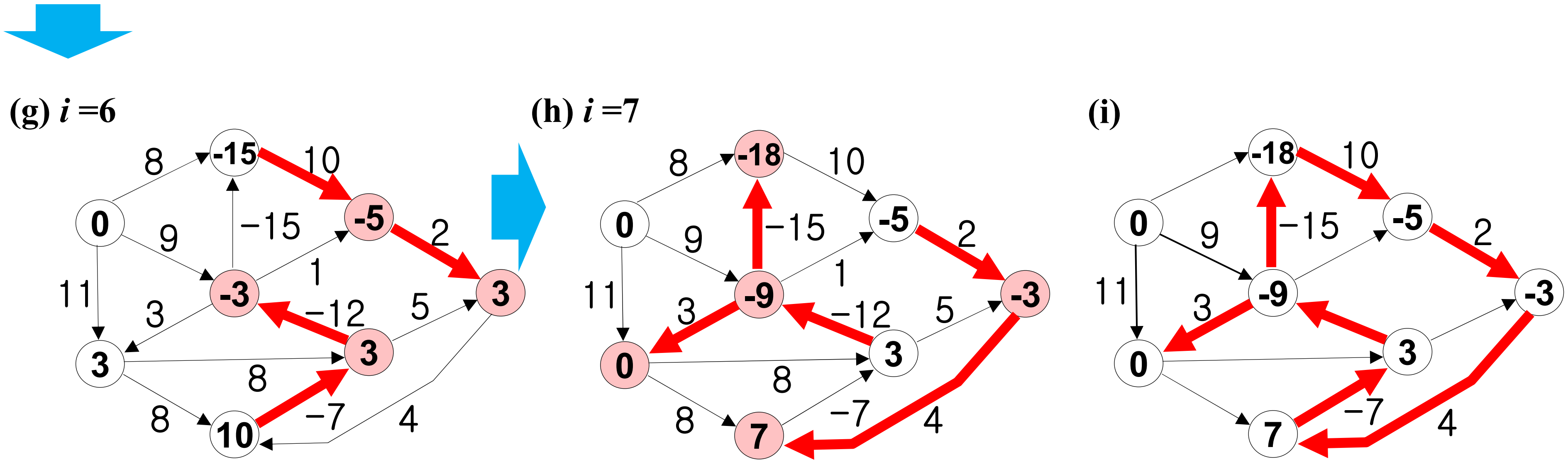
Bellman-Ford 알고리즘 (4/6)

- **Bellman-Ford 알고리즘:** 음의 싸이클이 있는 경우 #1



Bellman-Ford 알고리즘 (5/6)

- **Bellman-Ford 알고리즘:** 음의 싸이클이 있는 경우 #2



Bellman-Ford 알고리즘 (6/6)

- **Bellman-Ford 알고리즘: 동적 프로그래밍**
  - 동적 프로그래밍으로 본 벨만-포드 알고리즘
    - d_t^k: 중간에 최대 k개의 간선를 거쳐 정점 r로부터 정점 t에 이르는 최단거리
    - 목표: d_t^{n-1}

✓ 재귀적 관계

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

최소 신장 트리

단일 시작점 최단 경로:

DAG 최단 경로

DAG 최단 경로 (1/4)

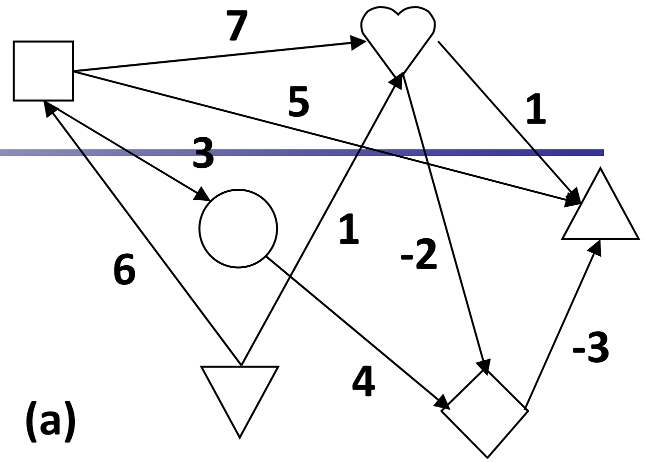
DAG 최단 경로

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

```
// DAG 최단 경로 알고리즘
DAG-ShortestPath(G, r):
        for each u∈V
                du ← ∞;
        dr ← 0;
        G의 정점들을 위상 정렬한다;
        for each u∈V (위상 정렬 순서로)
                for each v∈L(u)      ▷ L(u) : 정점 u로부터 연결된 정점들의 집합
                        if (du + wu,v < dv ) then dv ← du + wu,v ;
```

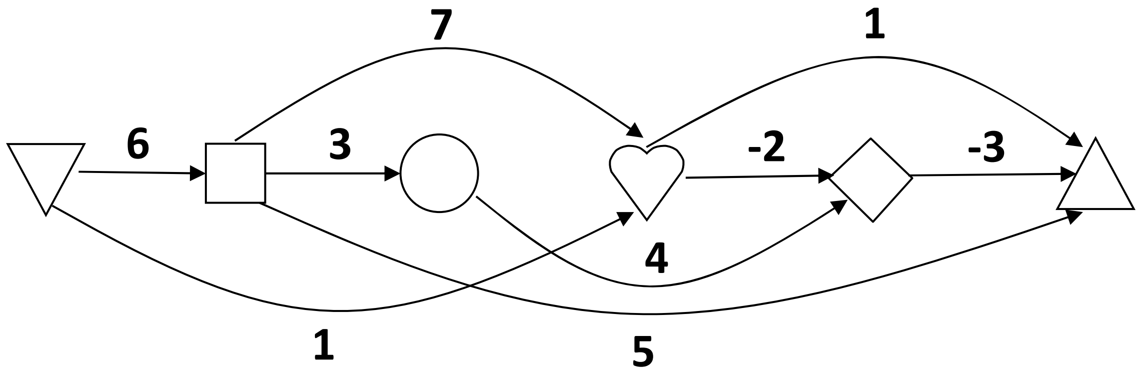
DAG 최단 경로 (2/4)

- **DAG 최단 경로:** 동작 과정 #1

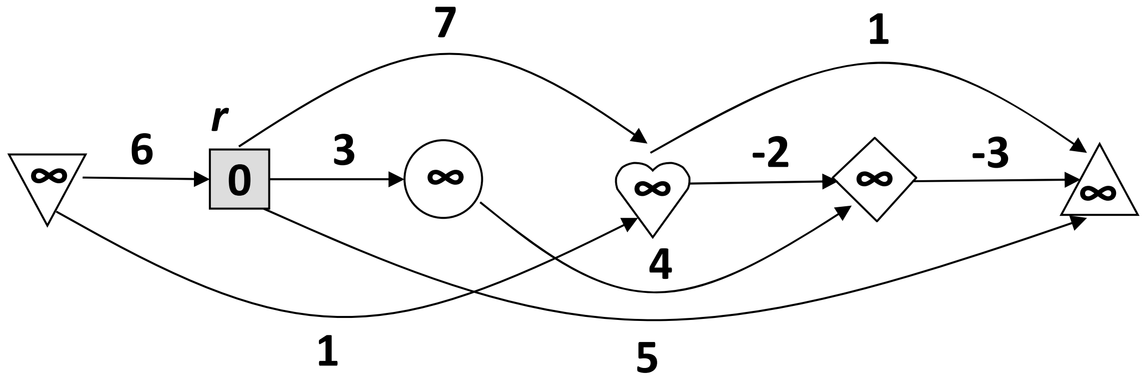


(a)

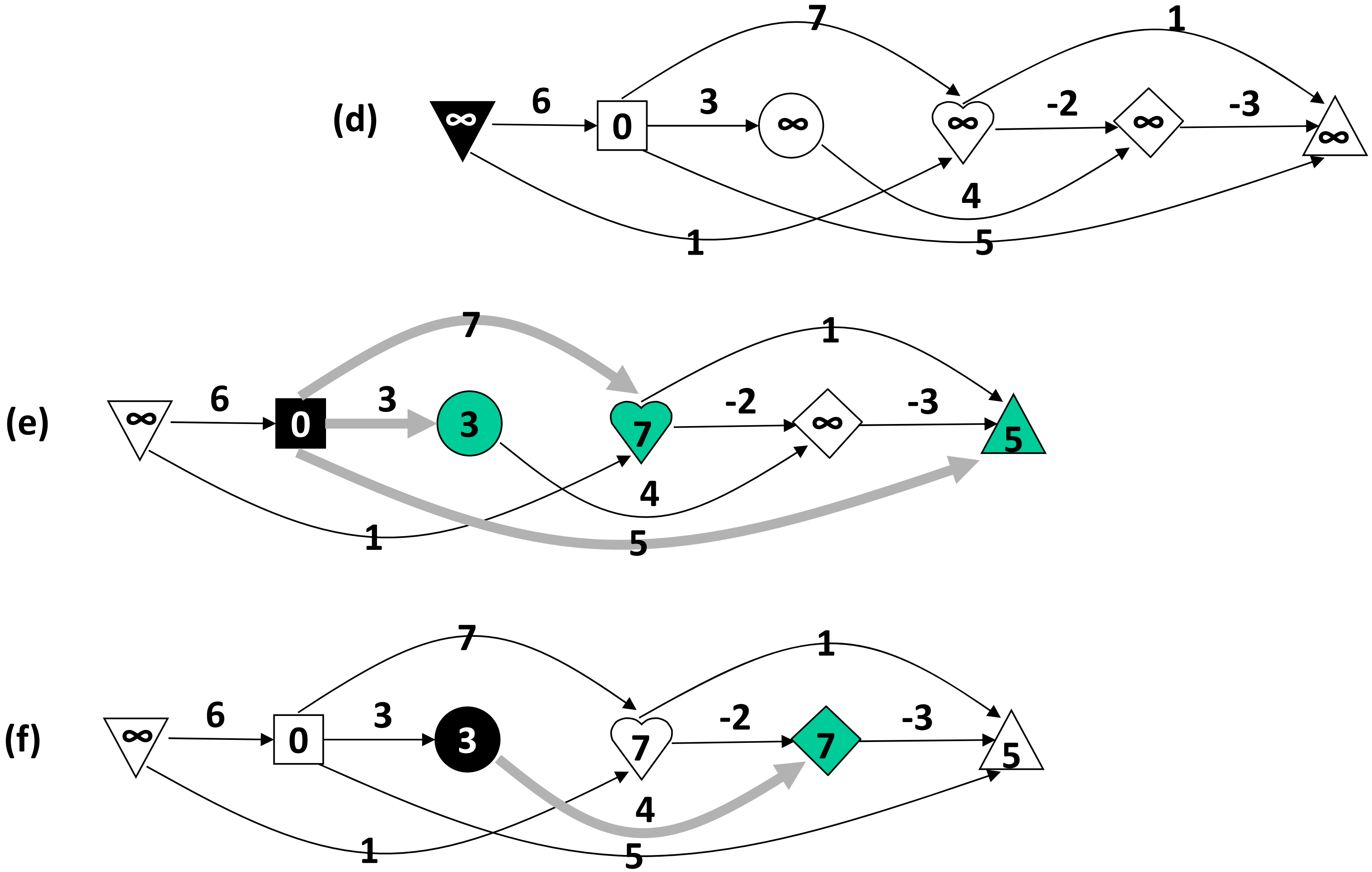
(b)



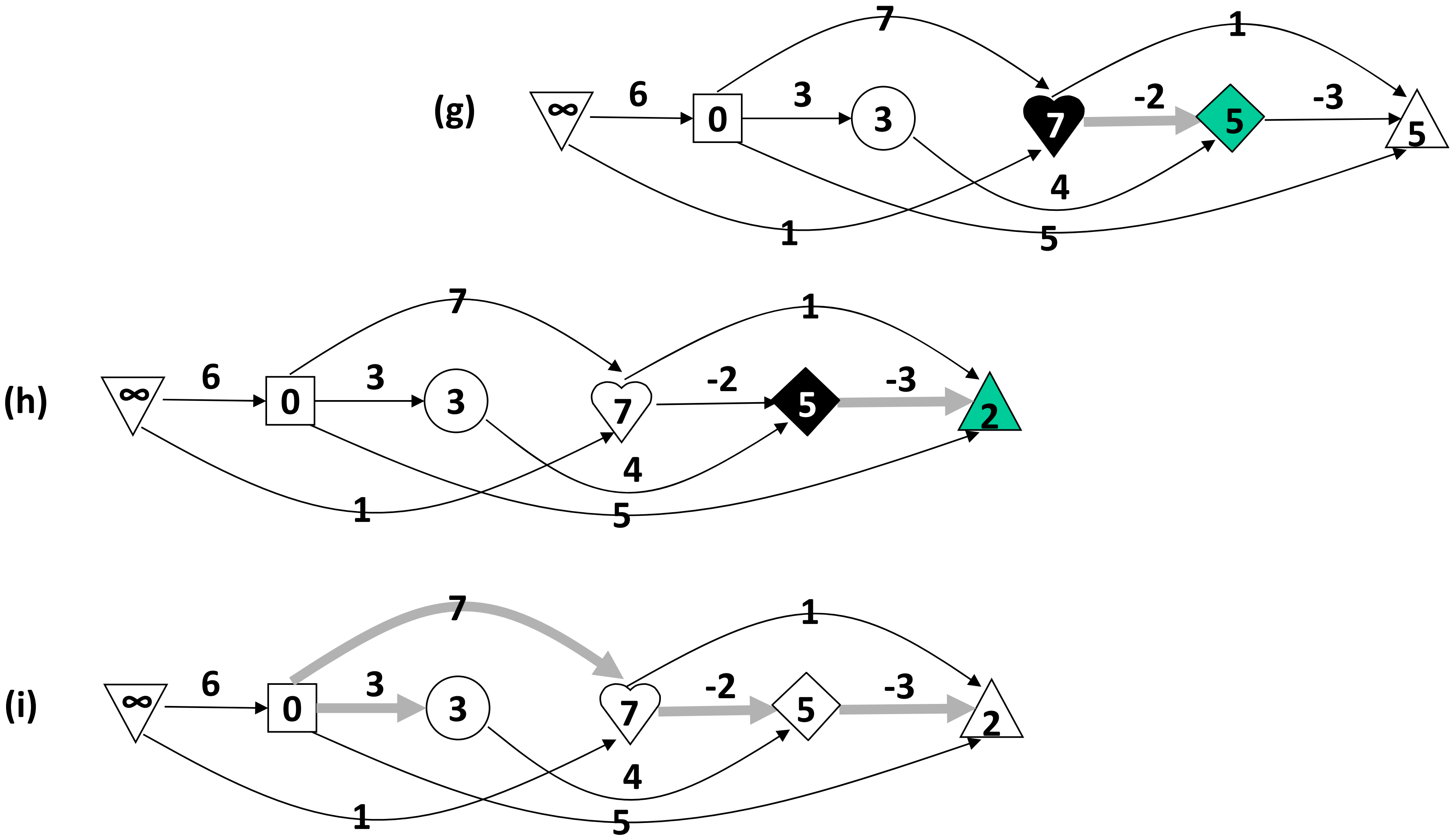
(c)



DAG 최단 경로 (3/4)



DAG 최단 경로 (4/4)



최소 신장 트리

모든 쌍 최단 경로:

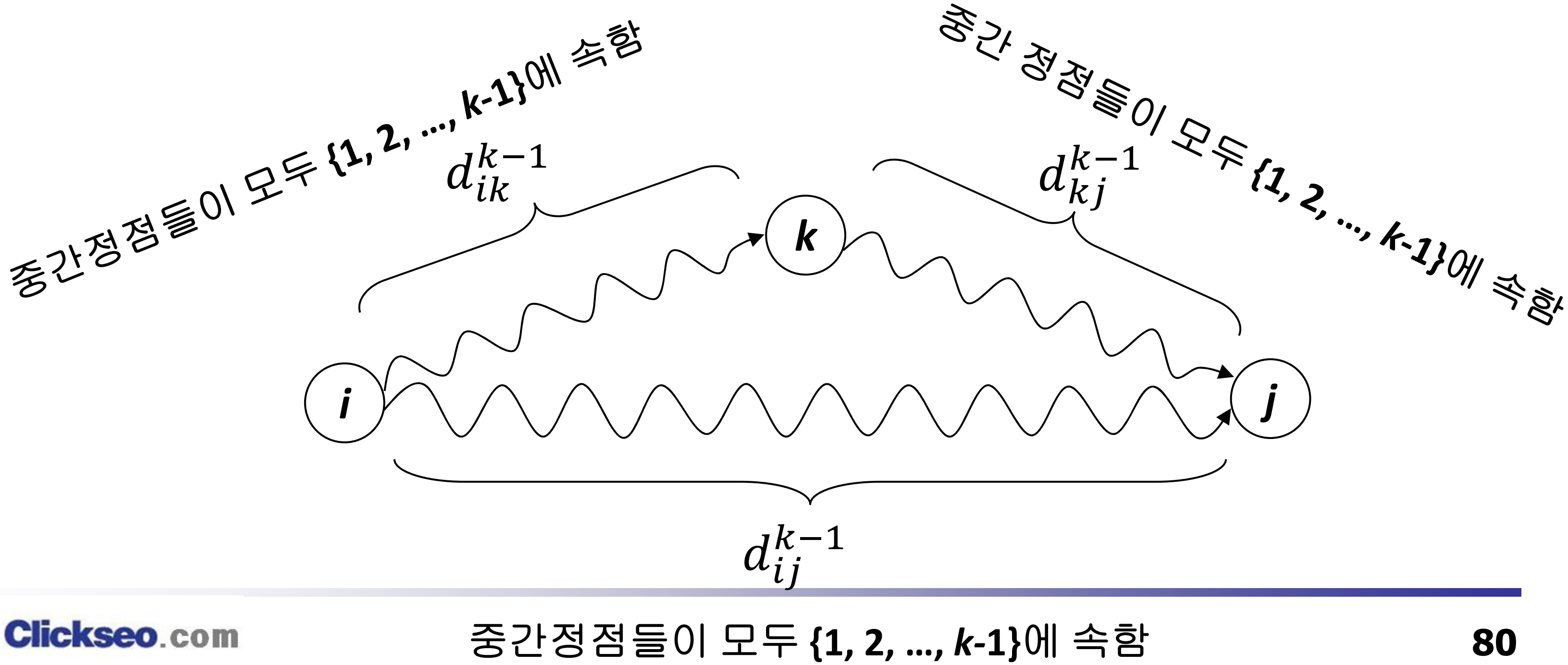
Floyd-Warshall 알고리즘

Floyd-Warshall 알고리즘 (1/2)

- **Floyd-Warshall 알고리즘**
  - 모든 정점들 간의 상호 최단거리 구하기
    - 응용 예: Road Atlas, 네비게이션 시스템, 네트워크 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 알고리즘 (2/2)

- **Floyd-Warshall 알고리즘:** 알고리즘

```
// FloydWarshall 알고리즘
FloydWarshall(G):
    for i ← 1 to n
        for j ← 1 to n
            d⁰ij ← wij;
    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
                dᵏij ← min {d⁰ij , dᵏ⁻¹ik + dᵏ⁻¹kj};
```

참고문헌

[1] Michael T. Goodrich 외 2인 지음, 김유성 외 2인 옮김, "C++로 구현하는 자료구조와 알고리즘", 한티에듀, 2020.

[2] 주우석, "IT CookBook, C · C++ 로 배우는 자료구조론", 한빛아카데미, 2019.

[3] 이지영, "C 로 배우는 쉬운 자료구조", 한빛아카데미, 2022.

[4] 문병로, "IT CookBook, 쉽게 배우는 알고리즘: 관계 중심의 사고법"(개정판), 개정판, 한빛아카데미, 2018.

[5] Richard E. Neapolitan, 도경구 역, "알고리즘 기초", 도서출판 홍릉, 2017.

[6] "프로그래밍 대회 공략을 위한 알고리즘과 자료 구조 입문", 와타노베 유타카 저, 윤인성 역, 인사이트, 2021.

[7] "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 문병로, 한빛아카데미, 2022.

[8] "이것이 취업을 위한 코딩 테스트다 with 파이썬", 나동빈, 한빛미디어, 2020.