2021-2
알고리즘

그래프 알고리즘II

한남대학교 컴퓨터공학과

그래프 알고리즘(11~12장) 구성

- **트리 순회하기**

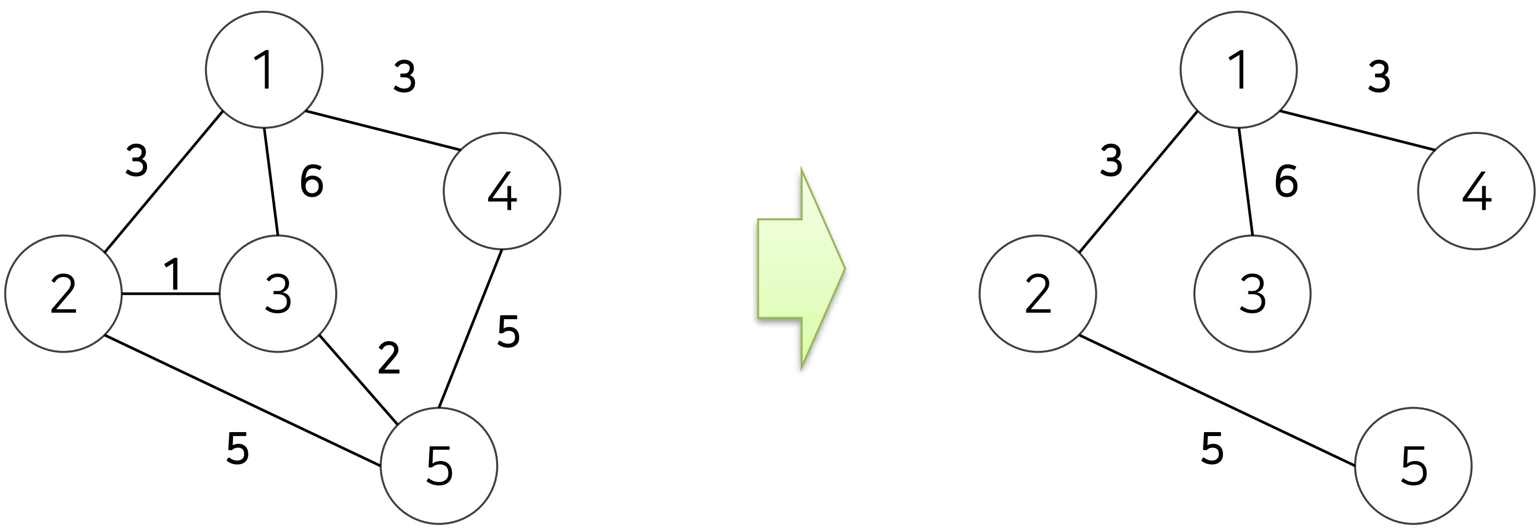
- **그래프**
  - 그래프란?
  - 그래프의 특징에 따른 분류
  - 그래프의 표현
  - nextworkx

- **그래프 탐색 알고리즘**
  - DFS, BFS
  - 최소신장트리
  - 최단 경로 알고리즘
  - 위상 정렬

최소신장트리

신장트리

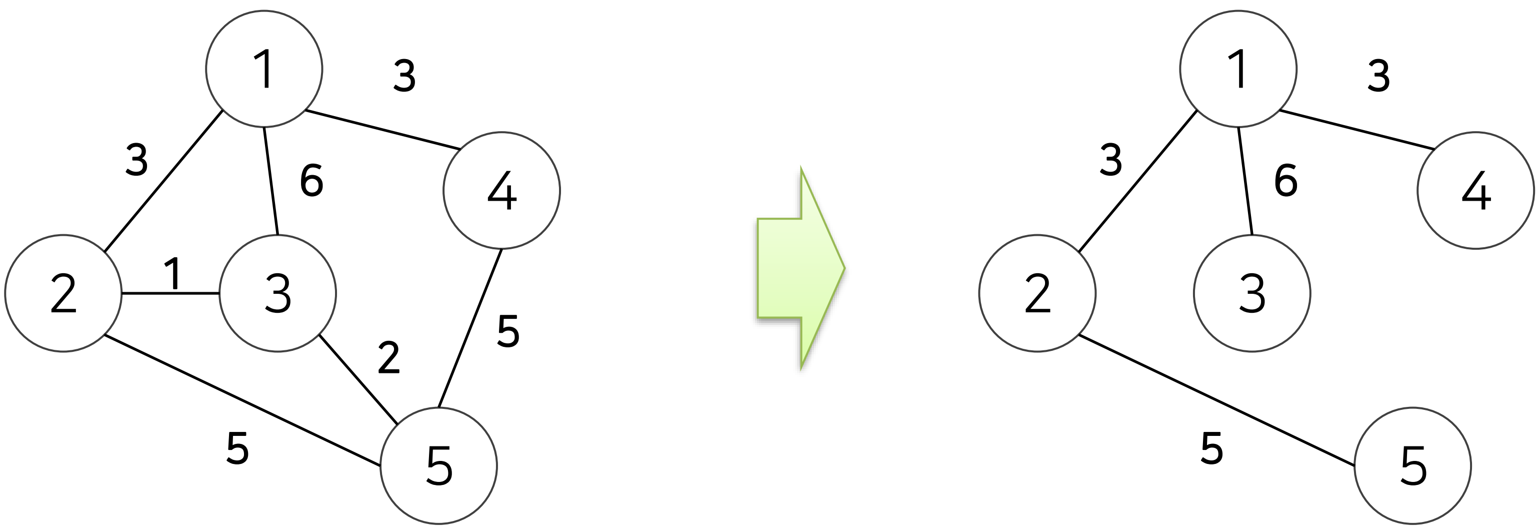
- **신장트리 Spanning Tree**
  - **(가중치가 있는) 무향 연결 그래프** G에 대해, 그래프 G의 신장트리는 *G*의 정점들과 간선들로만 구성된 트리
    - 싸이클 없는 연결 그래프
    - *n* 개의 정점을 가진 트리는 항상 *n*-1개의 간선을 갖는다



|V|=5, |E|=4

연습문제

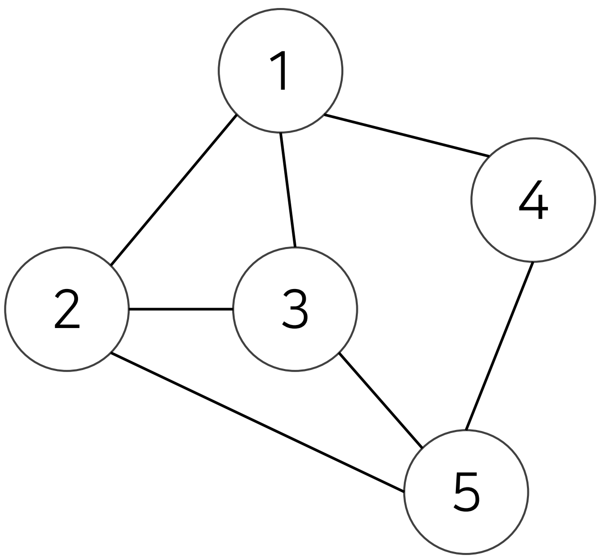
- **하나의 그래프에 대해 신장트리는 여러 개가 존재한다.**
  - 앞의 그래프에서 다른 신장 트리를 찾아 보자.
  - 선택한 간선들의 가중치 합은 얼마인가?



$|V|=5,\ |E|=4,\ \Sigma w=17$

연습문제

- **앞의 그래프에서 가중치를 삭제하였다. 이 그래프에서 임의의 신장트리 1개를 찾는 알고리즘을 작성해 보자.**
  - 트리는 <부모 – 자식> 형태로 출력
  - **1) DFS로 탐색**
  - **2) BFS로 탐색**



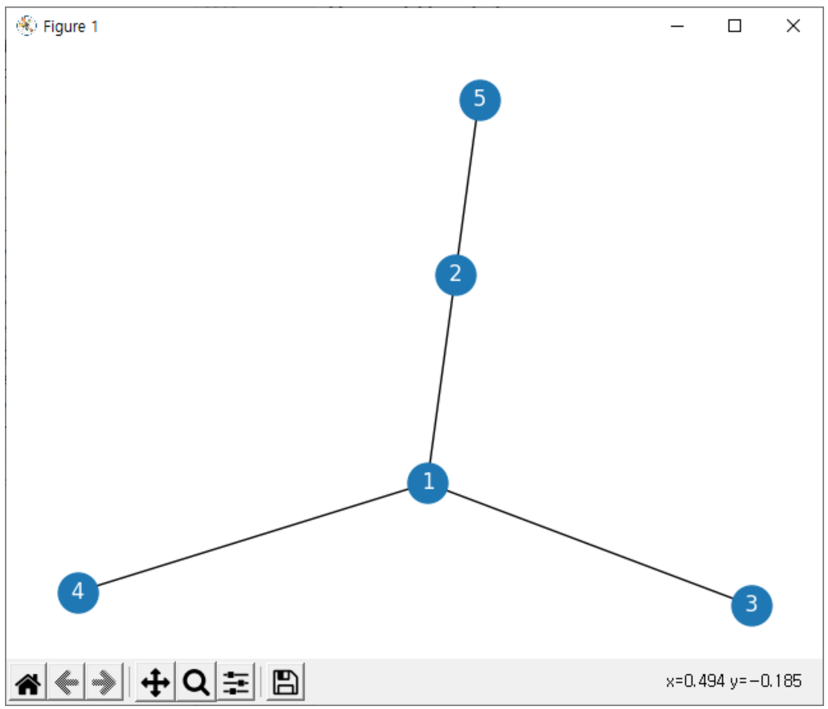

```
V = {1, 2, 3, 4, 5}
E ={(1, 2), (1, 3), (1, 4), (2, 3),
(2, 5), (3, 5), (4, 5)}
G = (V, E)

L = {v:[] for v in V}
for x, y in E:
    L[x].append(y)
    L[y].append(x)
```

```
1
1 - 2
2 - 3
3 - 5
5 - 4
```

```
1
1 - 2
1 - 3
1 - 4
2 - 5
```

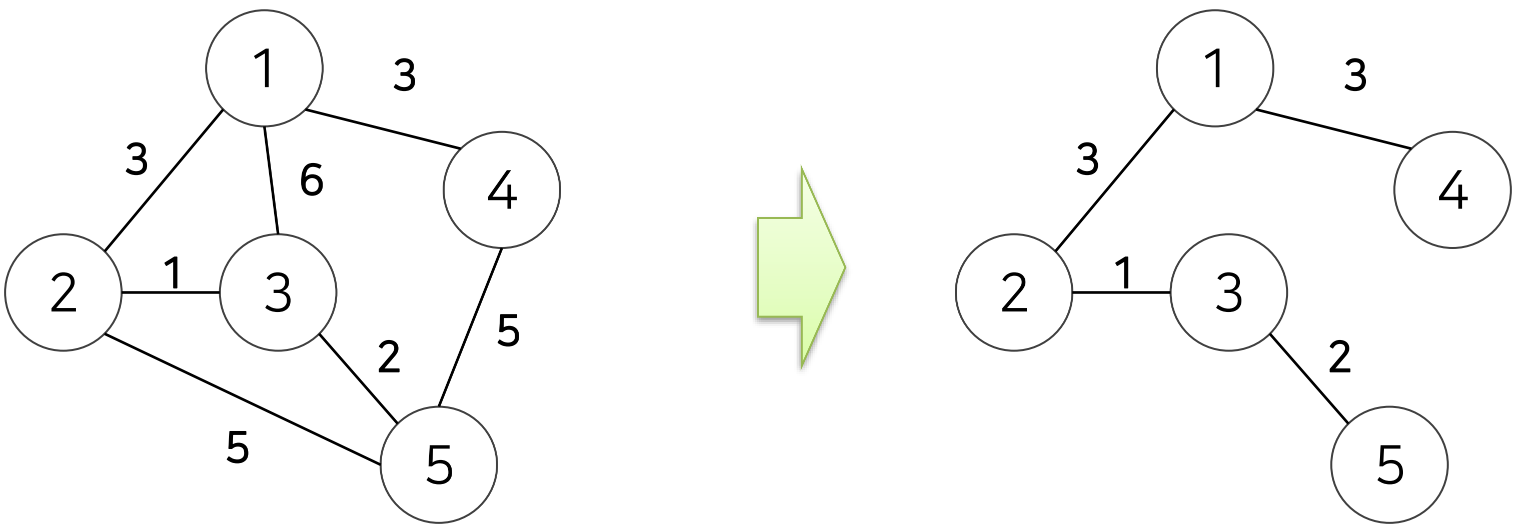
연습문제

- **다 한 사람은…**
  - *) 만들어진 트리를 networkx로 트리를 그려 본다.
  - *) 만들 수 있는 신장트리가 몇 종류인지 출력해 본다.



최소신장트리

- **최소신장트리(MST, Minimum Spanning Tree)**
  - 신장트리 중 간선들의 가중치 합이 가장 작은 트리
  - MST를 찾는 알고리즘?
  - 최소신장트리는 어떤 경우에 응용될까?



$|V|=5, |E|=4, \Sigma w=9$

최소신장트리

- MST도 여러 가지가 있을 수 있다.
  - 입력: 가중치가 있는 무향 연결 그래프
  - 출력: 최소신장트리(의 가중치 합)

  - 크루스칼 알고리즘 (Kruskal Algorithm)
  - 프림 알고리즘 (Prim Algorithm)

크루스칼 알고리즘

크루스칼 알고리즘

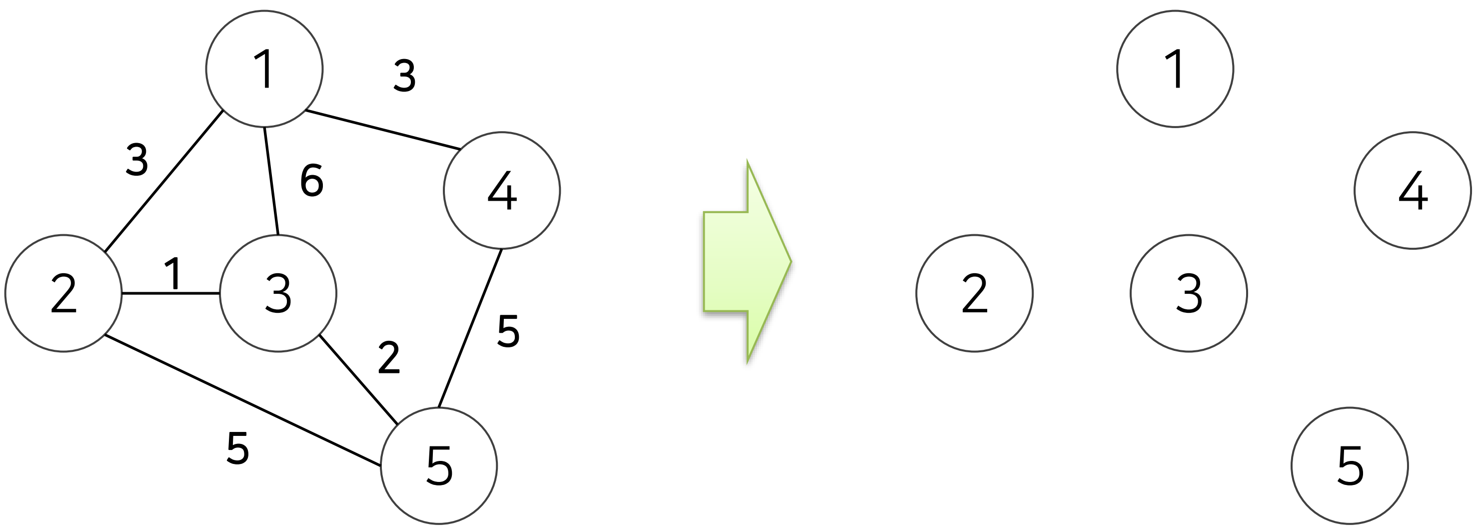
- **전체 간선들 중, 가중치가 낮은 순서대로 (V-1)개를 MST에 포함시킨다.**
  - 그리디Greedy 알고리즘의 일종

- **크루스칼 알고리즘(미완성)**

```
Kruskal(G=(V, E)) {
        E의 간선들을 가중치 오름차순으로 정렬;
        mst = {};
        (¦V¦-1)번 반복 {
                e <- E.pop_first();
                mst <- e;
        }
        return mst;
}
```

연습문제

- **크루스칼 알고리즘(미완성)**
  - 앞의 알고리즘을 아래 그래프에 적용해 보자.



크루스칼 알고리즘

- **그리디(Greedy) 알고리즘**
  - 앞을 내다보지 않고 **당장 눈앞에서 가장 좋아 보이는 선택만을 반복**하는 알고리즘
  - 프림 알고리즘, 크루스칼 알고리즘은 그리디 알고리즘으로 최적해를 구하는 드문 예

```
do {

      우선 가장 좋아 보이는 선택을 한다;

} until (해 구성이 끝남)
```

크루스칼 알고리즘

- **[안정성 정리]**
  - 프림과 크루스칼 알고리즘의 이론적 근거
  - 그래프의 정점들을 두 집합으로 나눴을 때,
  - **두 집합을 잇는 최소 간선이 있다면**
  - **그 간선을 포함하는 MST가**
  - **반드시 하나 이상 존재한다.**
  - 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}.

연습문제

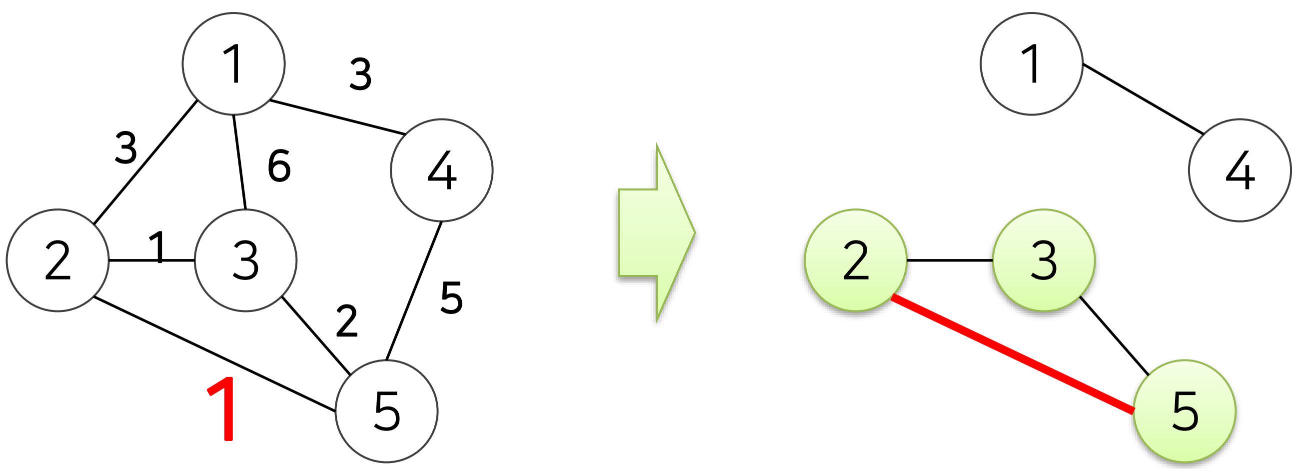
- **아래와 같이 간선에 가중치를 포함시킨 후, 크루스칼 알고리즘 (미완성)을 작성해 본다.**
  - MST에 포함된 간선들을 출력한다.

```
V = {1, 2, 3, 4, 5}
E = {(1, 2, 3), (1, 3, 6), (1, 4, 3), (2, 3, 1), (2, 5, 5),
(3, 5, 2), (4, 5, 5)}
E |= {(y, x, weight) for (x, y, weight) in E}

G = (V, E)
L = {v:[] for v in V}
for x, y, _ in E:
    L[x].append(y)
    L[y].append(x)
```

```
{(2, 3, 1), (3, 5, 2),
(1, 2, 3), (1, 4, 3)}
```

크루스칼 알고리즘

- **크루스칼 알고리즘(미완성)**
  - 간선 (2, 5)의 가중치를 1로 바꾸고 실행해 보자.
  - **싸이클이 만들어진다!**
    - mst에 포함된 정점끼리 연결하면 싸이클이 생김




```
{(2, 3, 1), (3, 5, 2),
(1, 4, 3), (2, 5, 1)}
```

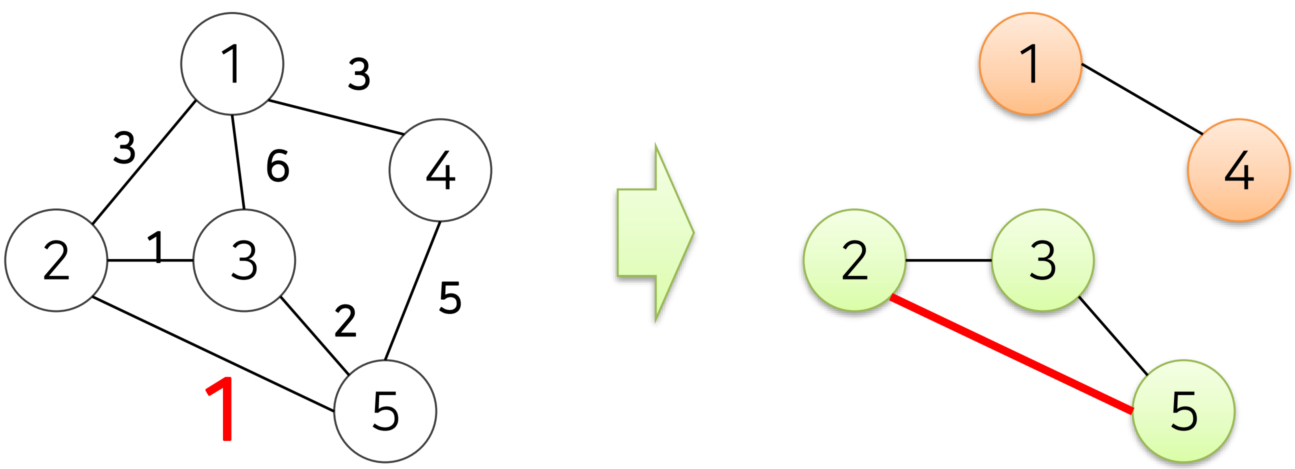
크루스칼 알고리즘

- **크루스칼 알고리즘(미완성v2)**
  - 간선들 중, 가중치가 낮은 순서대로 (V-1)개를 MST에 포함시킨다.
  - **간선의 정점이 둘 다 mst에 포함되어 있으면 무시한다:**
    - **생각대로 동작하지 않음**

```
Kruskal(G=(V, E)) {
        E의 간선들을 가중치 오름차순으로 정렬;

        mst_v, mst_e = {}, {};
        while ¦mst_e¦ < ¦V¦-1 {
                (v, w, weight) <- E.pop_first();
                if v in mst_v and w in mst_v: continue
                mst_v <- v; mst_v <- w;
                mst_e <- (v, w, weight);
        }
        return mst_e;
}
```

크루스칼 알고리즘

- **크루스칼 알고리즘(미완성v2)**
  - **MST를 만드는 과정에서 정점과 간선들은 몇 개의 집합을 만들게 된다.**
  - **간선은 두 집합을 하나로 합치는 역할**
  - ~~mst에 포함된 정점끼리 연결하면 싸이클이 생김~~
  - **같은 집합에 포함된 정점끼리 연결**하면 싸이클이 생김



{(2, 3, 1), (3, 5, 2),
(1, 4, 3), (2, 5, 1)}

크루스칼kruskal 알고리즘

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;

    정점 u와 정점 v가 서로 다른 집합에 속하면 {

        두 집합을 하나로 합친다;

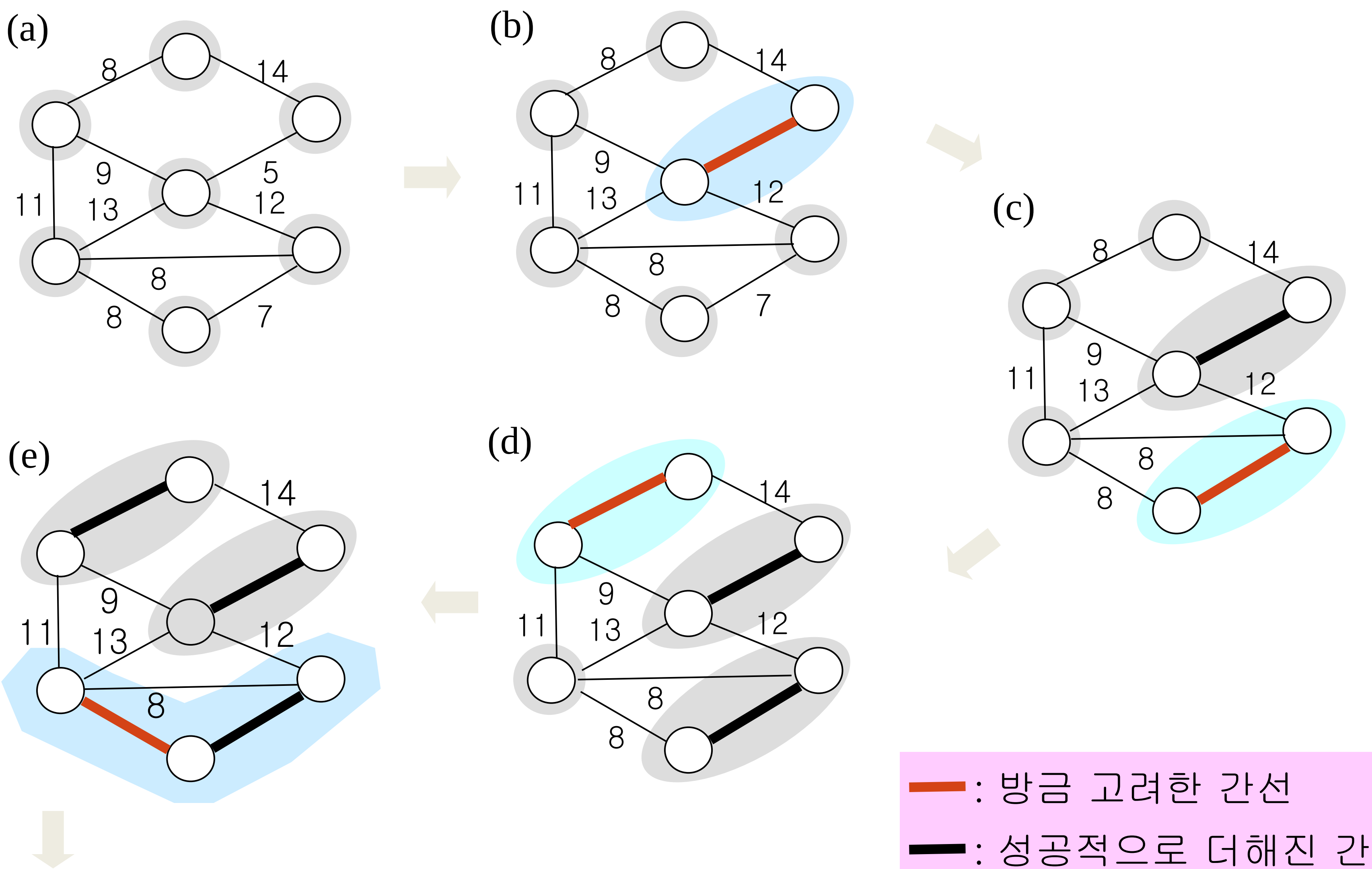
        $T \leftarrow T \cup \{(u, v)\}$;

    }

}

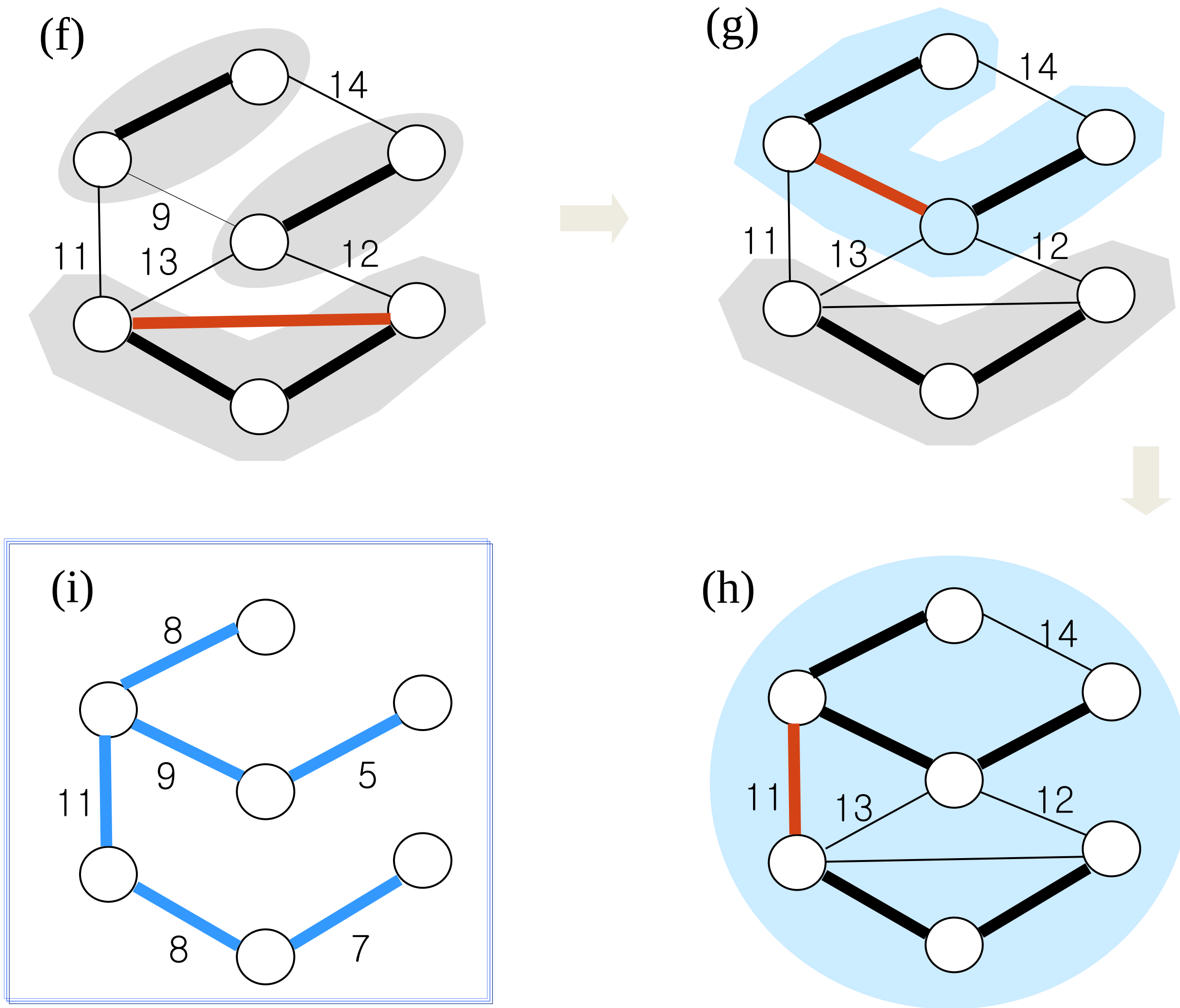
}

크루스칼 알고리즘의 작동 예



: 방금 고려한 간선

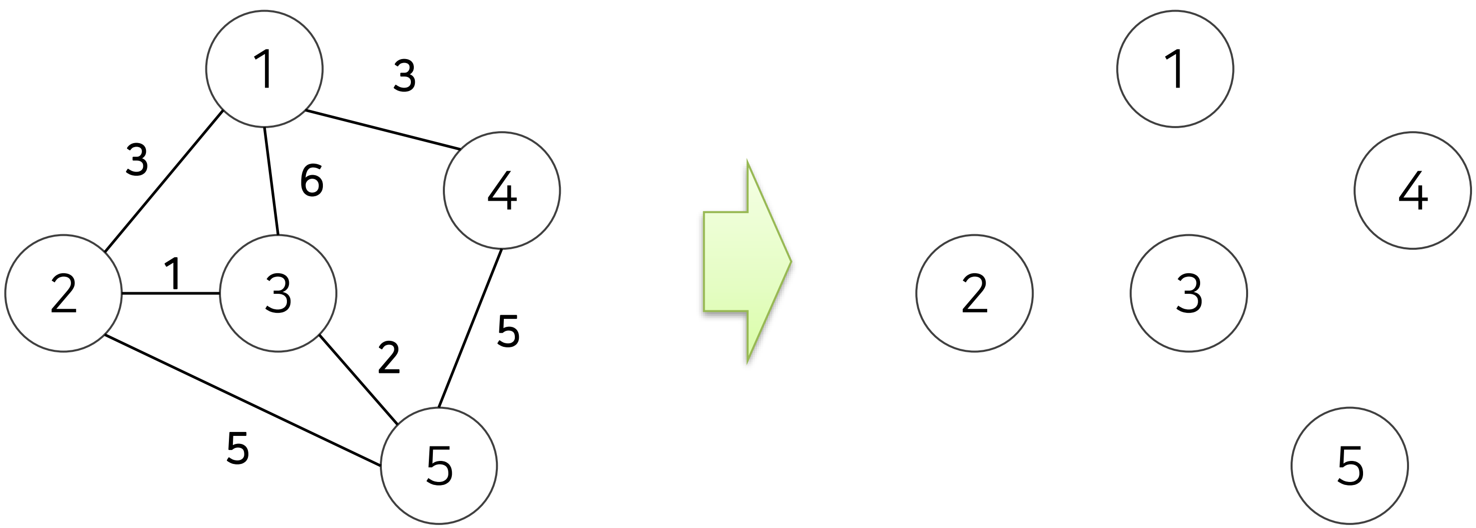
: 성공적으로 더해진 간선



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

연습문제

- **크루스칼 알고리즘**
  - 앞의 알고리즘을 아래 그래프에 적용해 보자.



크루스칼kruskal 알고리즘의 수행 시간

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;

    정점 u와 정점 v가 서로 다른 집합에 속하면 {

        두 집합을 하나로 합친다;

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

    }

}

}

Step 2: $\Theta(V)$

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

Loop 4: $O(E\log^* V)$

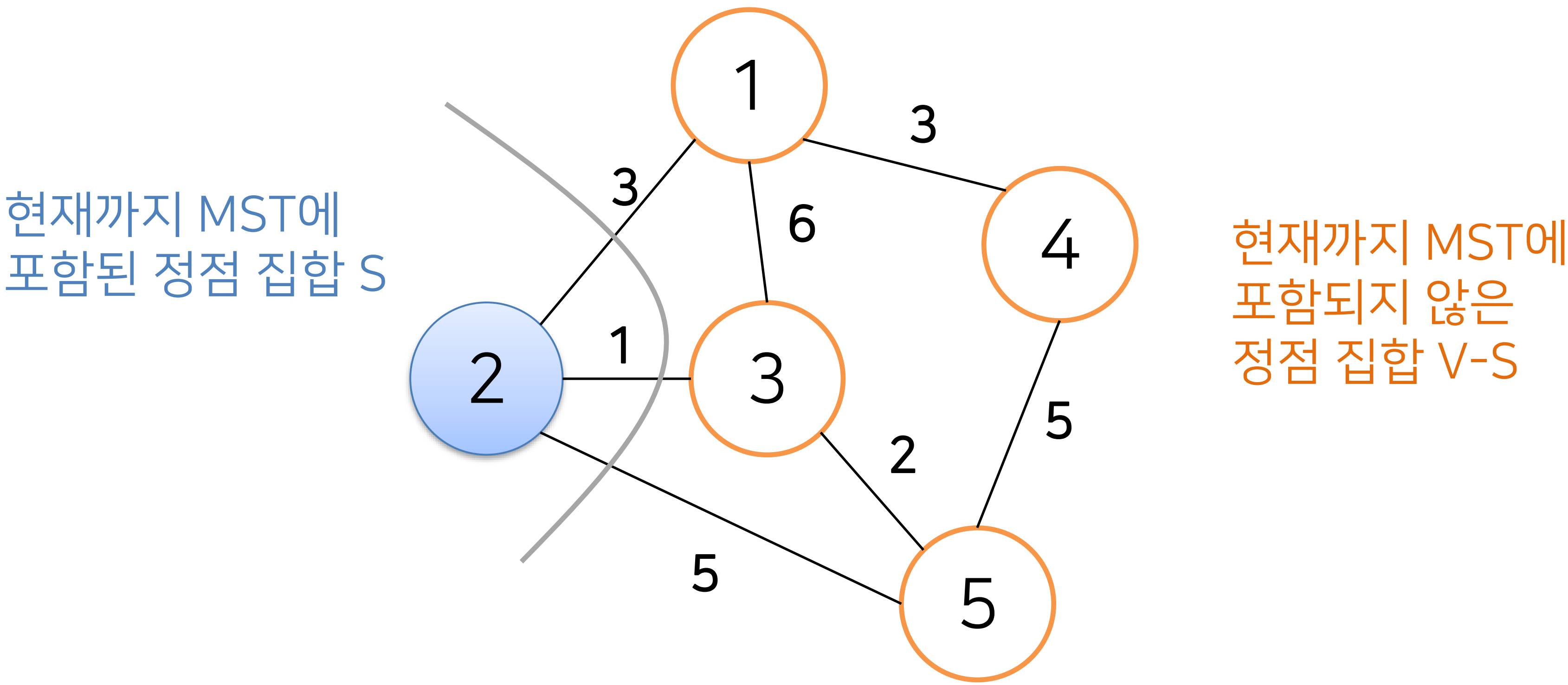
by an efficient set handling

✓수행시간: $O(E\log V)$

프림 알고리즘

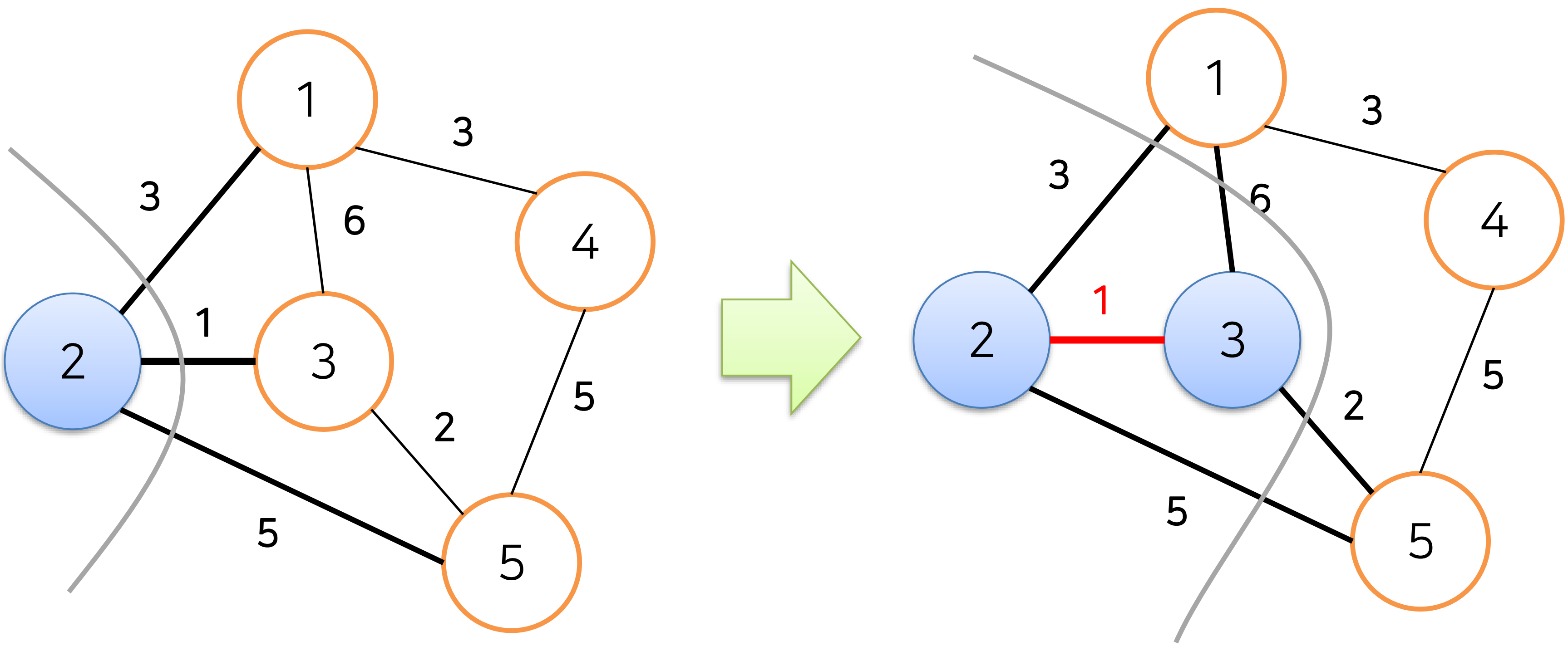
프림Prim 알고리즘

- **(|V| - 1) 번 반복:**
  – 현재까지 만들어진 MST에 (정점1 + 간선1)을 추가해서 MST를 확장한다.



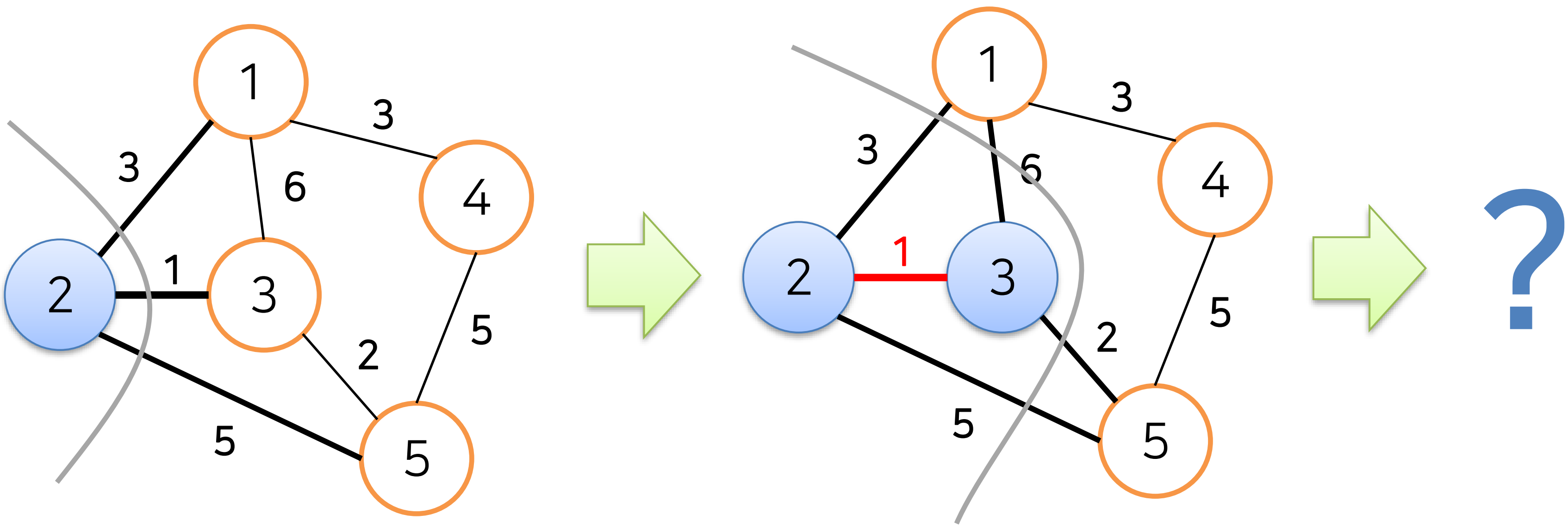
프림Prim 알고리즘

- **(|V| - 1) 번 반복:**
  - 현재까지 만들어진 MST에 (정점1 + 간선1)을 추가해서 MST를 확장한다.
  - 이 때 V와 V-S 를 잇는 최소 길이 간선을 선택한다.



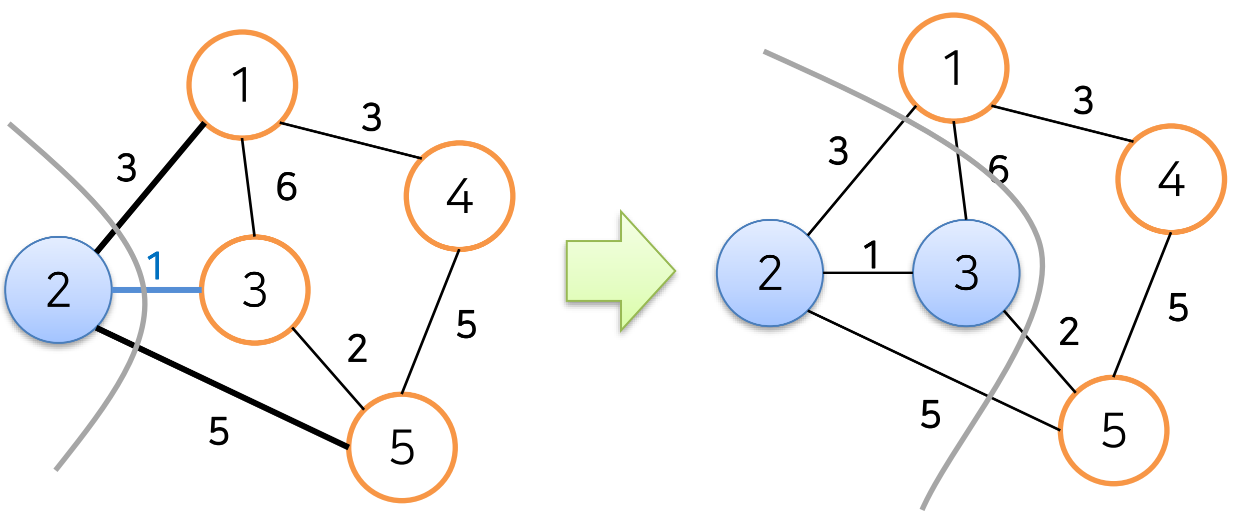
프림Prim 알고리즘

- **아래 그림에 이어서, 프림 알고리즘으로 최소신장트리를 완성해 보자.**
  - 2번 정점에서 시작한 경우 / 1번 정점에서 시작한 경우



프림Prim 알고리즘

- **프림 알고리즘 구현(미완성) –필요한 것들**
  - 1) MST에 속한 정점(S)과 남아 있는 정점(V-S)을 기억한다.
  - 2) 두 집합 S, V-S를 잇는 간선들을 찾고
  - 3) 그 중 가장 짧은 간선을 찾는다.
  - 4) 찾은 간선을 MST에 추가하고(출력)
  - 5) 간선에 연결된 두 정점 중 V-S에 속한 노드를 S로 옮긴다.



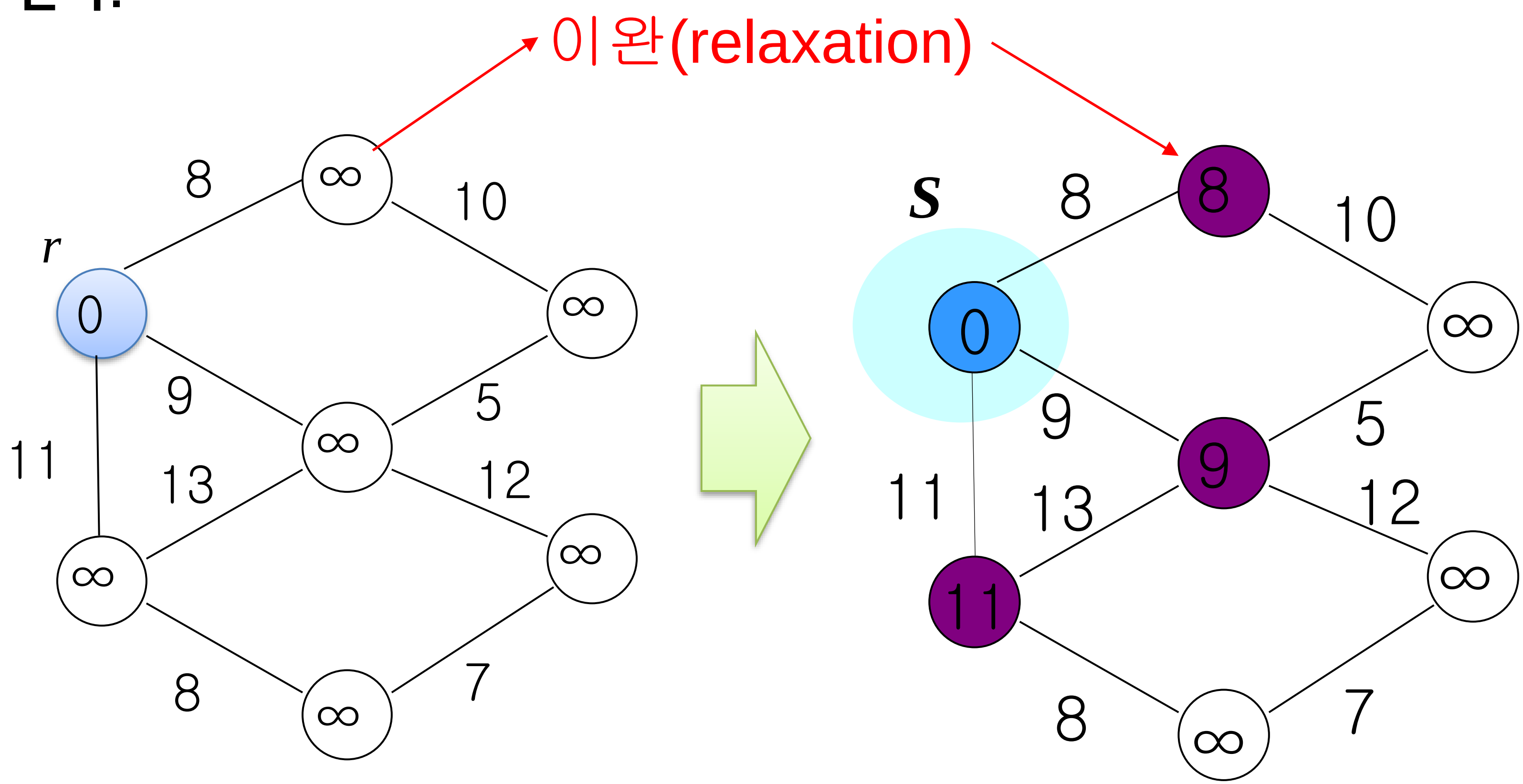
프림Prim 알고리즘

```
Prim (G, r)
{
      S ← Φ;
      정점 r을 방문되었다고 표시하고, 집합 S에 포함시킨다;

   while (S≠ V) {
         S에서 V-S를 연결하는 간선들 중
           최소길이의 간선 (x, y) 를 찾는다; ▷ (x∈S, y∈V-S)

         정점 y를 방문되었다고 표시하고, 집합 S에 포함시킨다;
   }
}
```

✓수행 시간: $O(|E|\log|V|)$ ← 힙 이용

프림 알고리즘을 좀 더 구체적으로

- S와 V-S를 연결하는 간선 중 최소길이의 간선을 찾는다.
- → **V-S에 속한 정점들 중 S와 인접한 정점**들을 **MST에 연결하는 비용**을 기억한다.



프림 알고리즘을 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 $(S \neq V)${ ▷ n**회 순환된다**
    $u \leftarrow$ **extractMin(V-S, d)** ;
    $S \leftarrow S \cup \{u\}$;
    for each $v \in L(u)$ ▷ $L(u)$: u**로부터 연결된 정점들의 집합**
      if ($v \in V$-S **and** $w_{uv} < d_v$) **then** $d_v \leftarrow w_{uv}$;
  }
}

d: V-S의 정점 하나를 S에 연결하는 비용

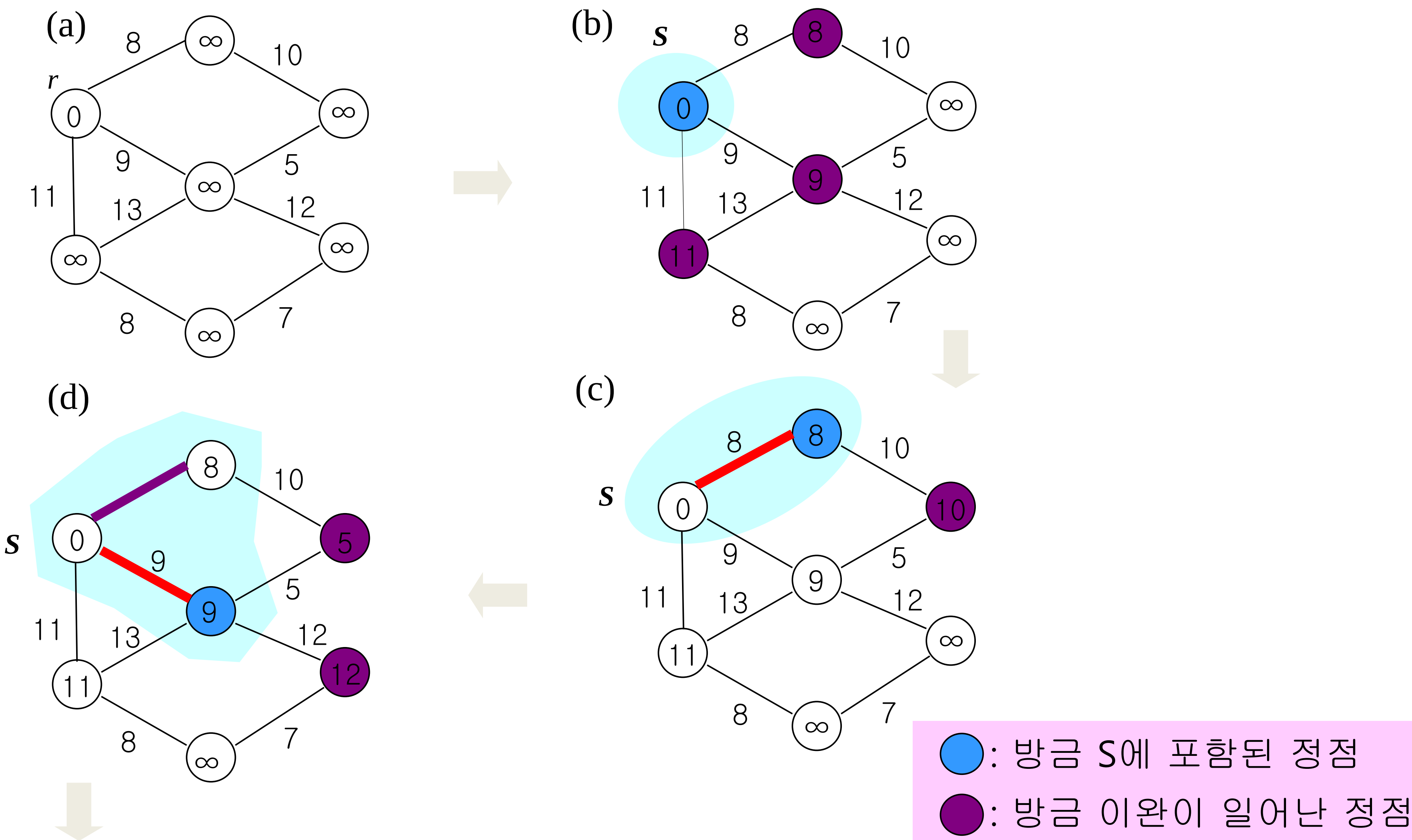
이완(relaxation)

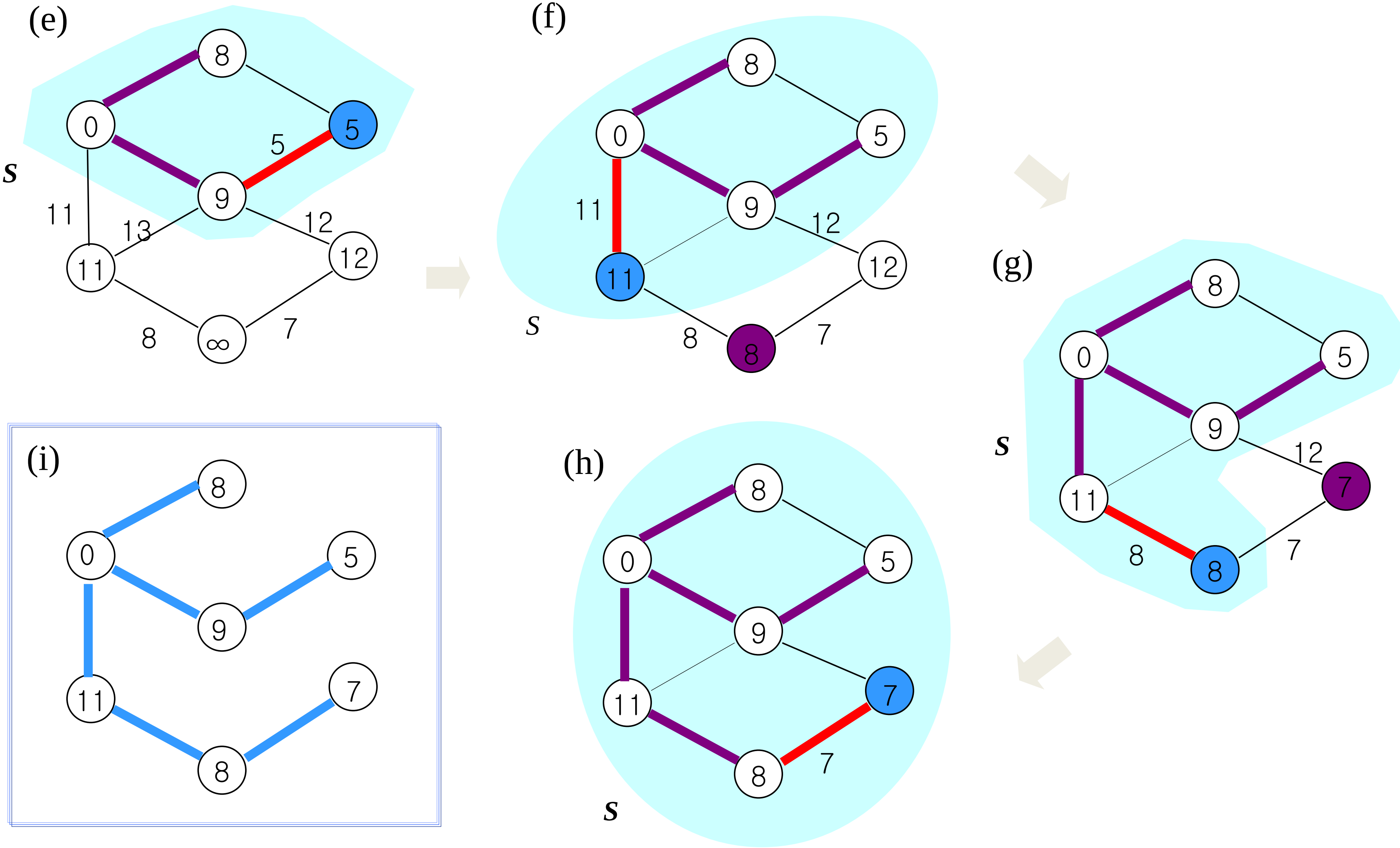
extractMin(Q, d)
{
  집합 Q**에서** d**값이 가장 작은 정점** u**를 리턴한다** ;
}

✓수행시간: $O(|E|\log|V|)$

힙 이용

프림 알고리즘의 작동 예




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

프림 알고리즘 - 구현

- **아래와 같이 간선에 가중치를 포함시킨다.**

```
V = {1, 2, 3, 4, 5}
E = {(1, 2):3, (1, 3):6, (1, 4):3, (2, 3):1,
(2, 5):5, (3, 5):2, (4, 5):5}
E.update({(y, x): E[(x, y)] for (x, y) in E})

WEIGHT_MAX = 1000
G = (V, E)
L = {v:set() for v in V}
for x, y in E:
    L[x].add(y)
    L[y].add(x)
```

프림 알고리즘 - 구현

- **초기화**
  - **S** : 방문한(MST에 포함된) 정점 집합
  - **D[x]** : 정점 x를 MST에 연결하는 비용

```
def prim(s):
    D = {v:WEIGHT_MAX for v in V}
    D[s] = 0
    S = set()
    # print(D)
```

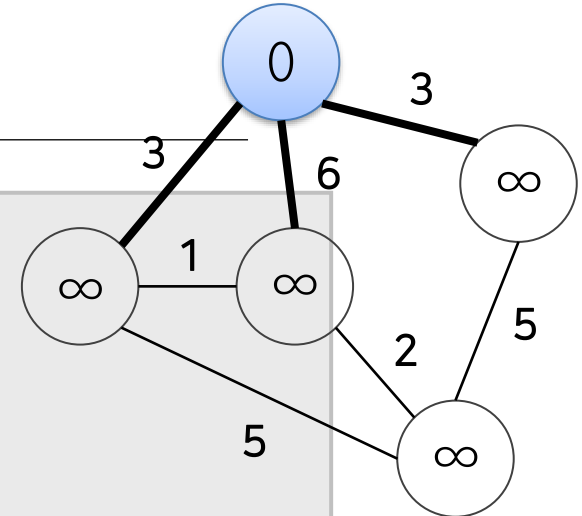
프림 알고리즘 - 구현

- **S와 V-S를 연결하는 최소 가중치 간선(정점)을 찾는다.**
  – **extractMin()**을 작성해 보자.

```
def extractMin(Q, D):
        pass  # 정점 집합 Q에서 D[x]가 최소인 x를 리턴

def prim(s):
    ...  # ← 앞 장의 초기화 코드
    while S != V:
        # 최소 비용 노드 u를 방문한다(MST에 포함시킴).
        # 힙으로 구현 가능
        u = extractMin(V - S, D)
        S.add(u)

    return sum(D.values())

print('(prim) MST =', prim(1))
```

```
(prim) MST = 4000
```

프림 알고리즘 - 구현




```
def prim(s):
    D = {v:WEIGHT_MAX for v in V}
    D[s] = 0
    # print(D)

    S = set()
    while S != V:
        u = extractMin(V - S, D)
        S.add(u)
```

u와 이웃한 정점 중 S에 속하지 않는 정점 v들에 대해,
MST에 연결하는 최소 비용을 업데이트한다.
(간선 (u, v)의 가중치와 D[v]를 비교)

```
    return sum(D.values())
```

```
(prim) MST = 9
```

최단경로 알고리즘

최단경로 문제 Shortest Paths

- **입력: 가중치가 있는 유향 그래프**
  - (무향 그래프는 유향 그래프로 변환할 수 있다)
- **가정**
  - 가중치의 합이 음수인 싸이클이 없음
  - → 문제가 성립하지 않는다.
- **출력: 두 정점 사이의 최단경로(의 길이)**
  - 두 정점 사이의 경로들 중
  - 간선의 가중치 합이 최소인 경로

최단경로 문제 Shortest Paths

- **단일 시작점 최단경로 문제**
  - 단일 시작점으로부터 각 정점에 이르는 최단경로를 구한다.

  - **다익스트라 알고리즘**
    - **음의 가중치를 허용하지 않는 최단경로**
  - **벨만-포드 알고리즘**
    - 음의 가중치를 허용하는 최단경로

- **모든 쌍 최단경로 문제**
  - 모든 정점 쌍 사이의 최단경로를 모두 구한다.

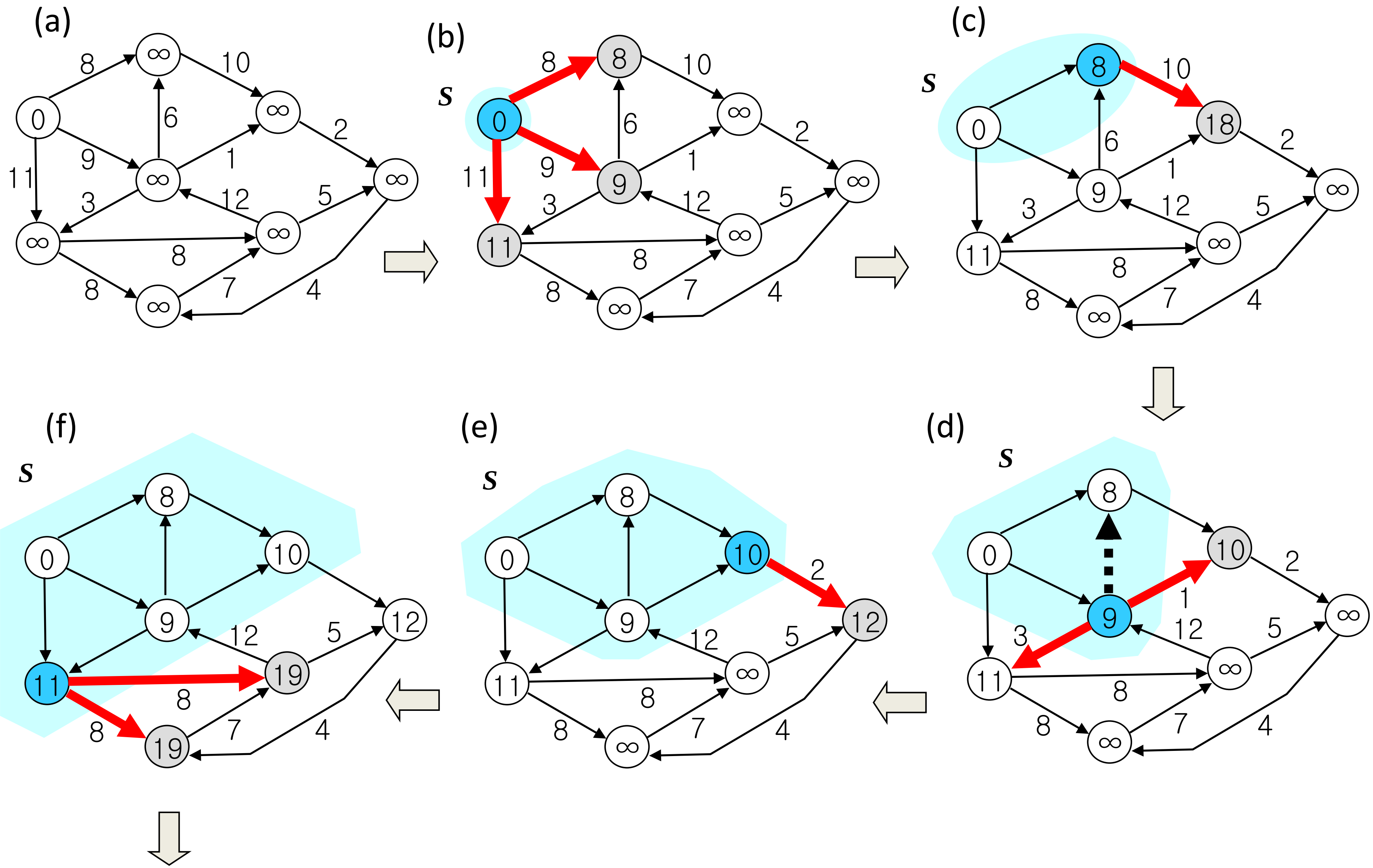
  - **플로이드-워샬 알고리즘**

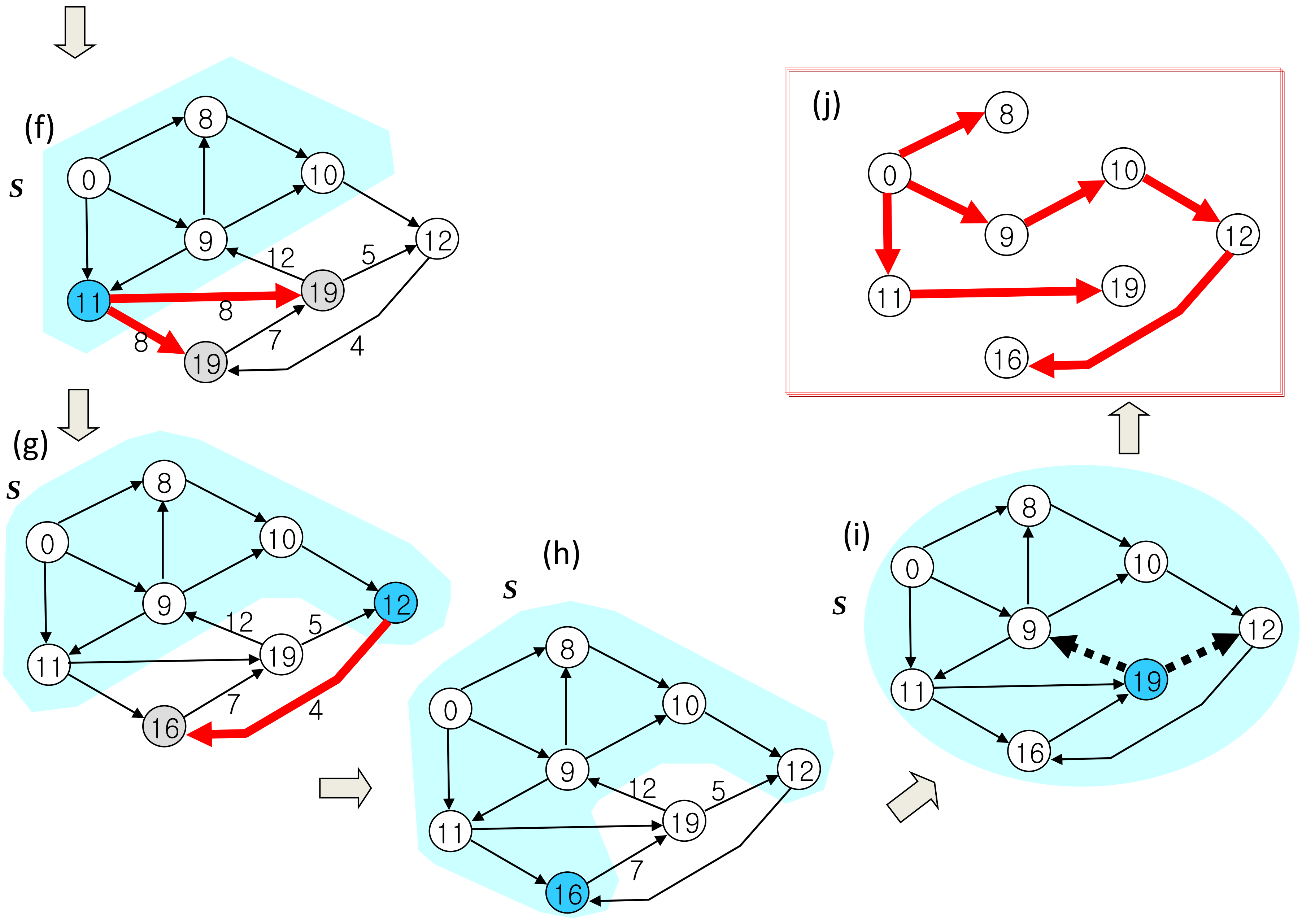
최단경로 문제 Shortest Paths

- **다익스트라 알고리즘**
  - 가중치가 있는 유향 그래프
  - 시작점으로부터 최단경로(음의 가중치를 허용하지 않음)

- **프림 알고리즘과 같은 구조**
  - MST에 정점 v를 연결하는 비용
  - → 시작점 s부터 v까지 오는 최단경로의 길이

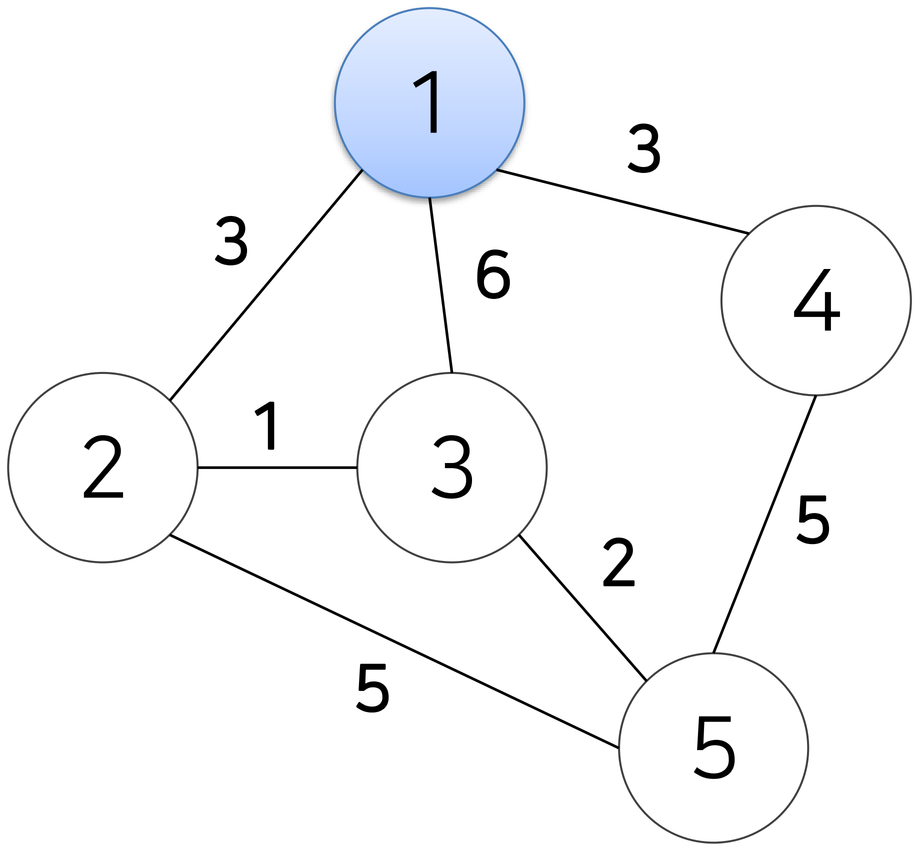
다익스트라 알고리즘의 작동 예




(f)
s
(g)
s
(h)
s
(i)
s
(j)

연습문제

- **다익스트라 알고리즘으로 정점 1에서 각 정점까지 도착하는 최단경로를 구해 보자.**



다익스트라Dijkstra 알고리즘

모든 간선의 가중치는 음이 아니어야 함

```
Dijkstra(G, r)
▷ G=(V, E): 주어진 그래프
▷ r: 시작으로 삼을 정점
{
     S ← Φ ;                         ▷ S : 정점 집합
     for each u∈V
          d[u] ← ∞ ;
     d[r] ← 0 ;
     while (S≠V){                    ▷ n회 순환된다
          u ← extractMin(V-S, d) ;
          S ← S ∪ {u};
          for each v∈L(u)            ▷ L(u) : u로부터 연결된 정점들의 집합
               if (v∈V-S and d[u] +w[u, v] < d[v] ) then {
                         d[v] ← d[u] + w[u, v];
                         prev[v] ← u;
     }
}

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

이완(relaxation)

✓수행 시간: $O(|E|\log|V|)$

힙 이용

다익스트라 알고리즘 - 구현

- 프림 알고리즘에서 거리 배열 D를 리턴하도록 수정한다.

```
def dijkstra(s):
    ...
    return D
```

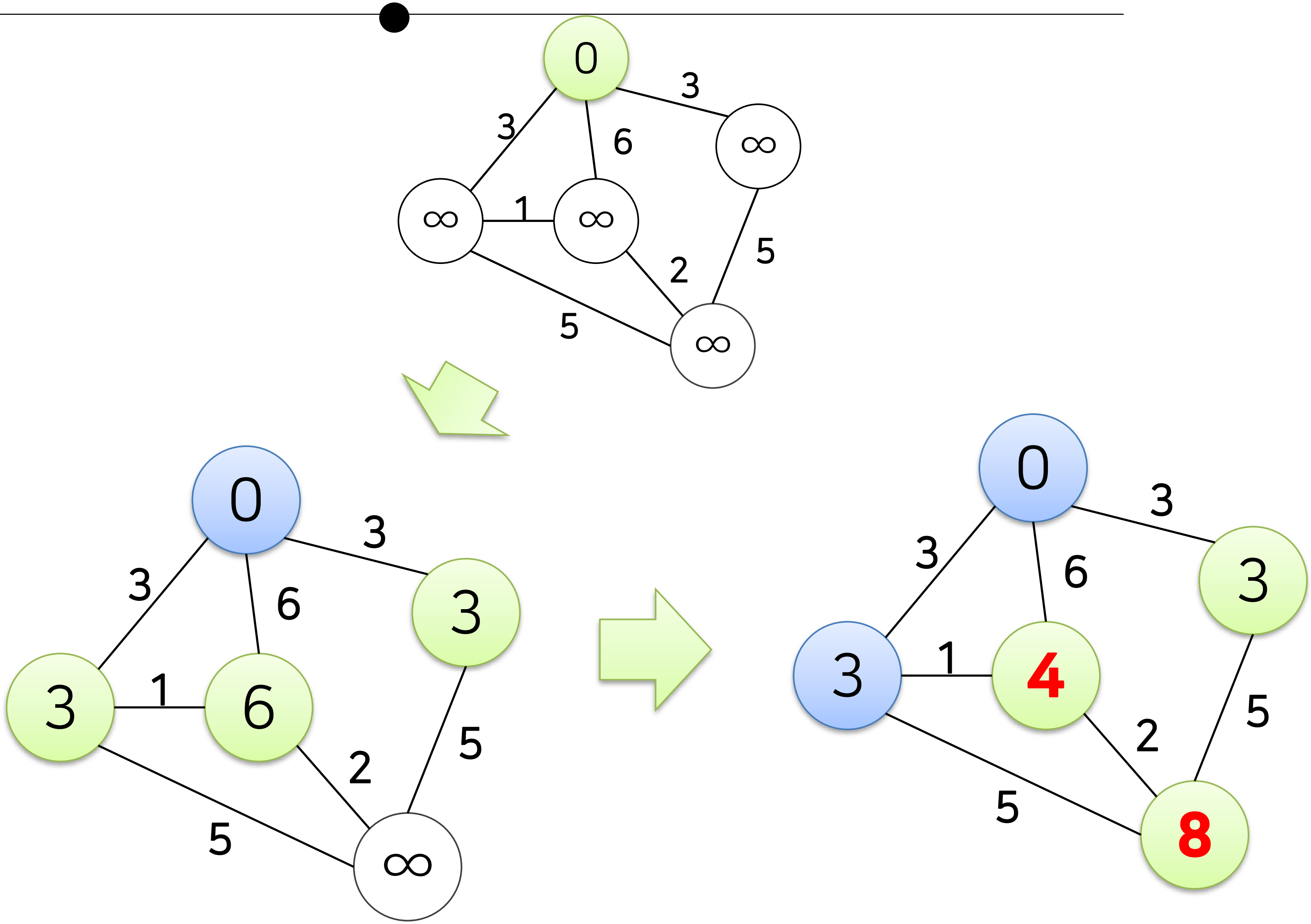
```
print('(dijkstra) Shorted Path Length')
spaths = dijkstra(1)
for v in V:
    print('1 ~>', v, ':', spaths[v])
```

다익스트라 알고리즘 - 구현

```
def dijkstra(s):
    D = {v:WEIGHT_MAX for v in V}
    D[s] = 0
    # print(D)

    S = set()
    while S != V:
        u = min(V - S, key=lambda x: D[x])
        S.add(u)
        for v in L[u] - S:
            D[v] =
            # print

    return D
```

현재까지 s~>v 최단경로 D[v]와 s~>u 최단경로 D[u]에서 (u, v)를 거쳐 v로 오는 거리를 비교

다익스트라 알고리즘 - 구현



벨만-포드Bellman-Ford 알고리즘

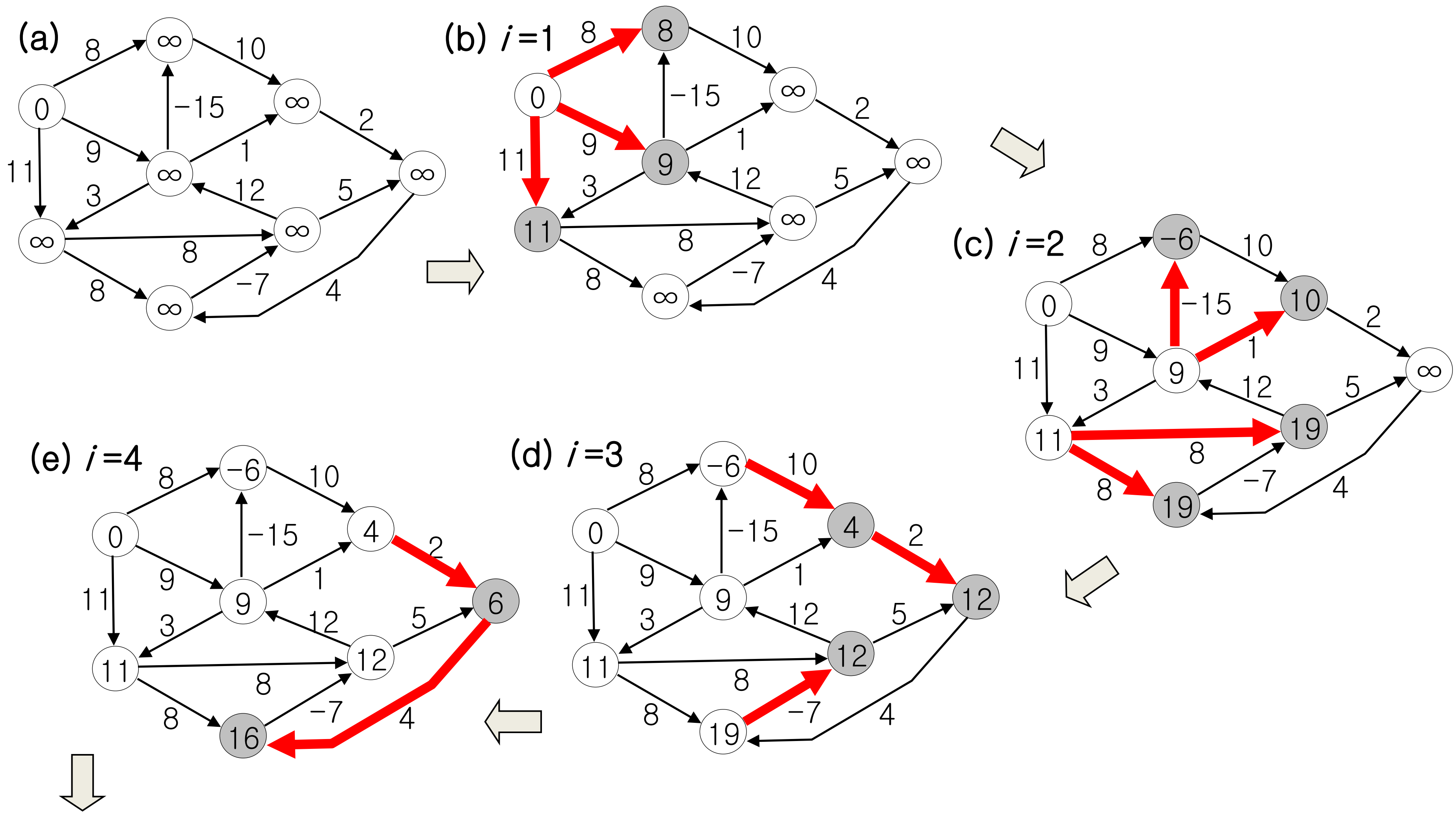
음의 가중치를 허용한다

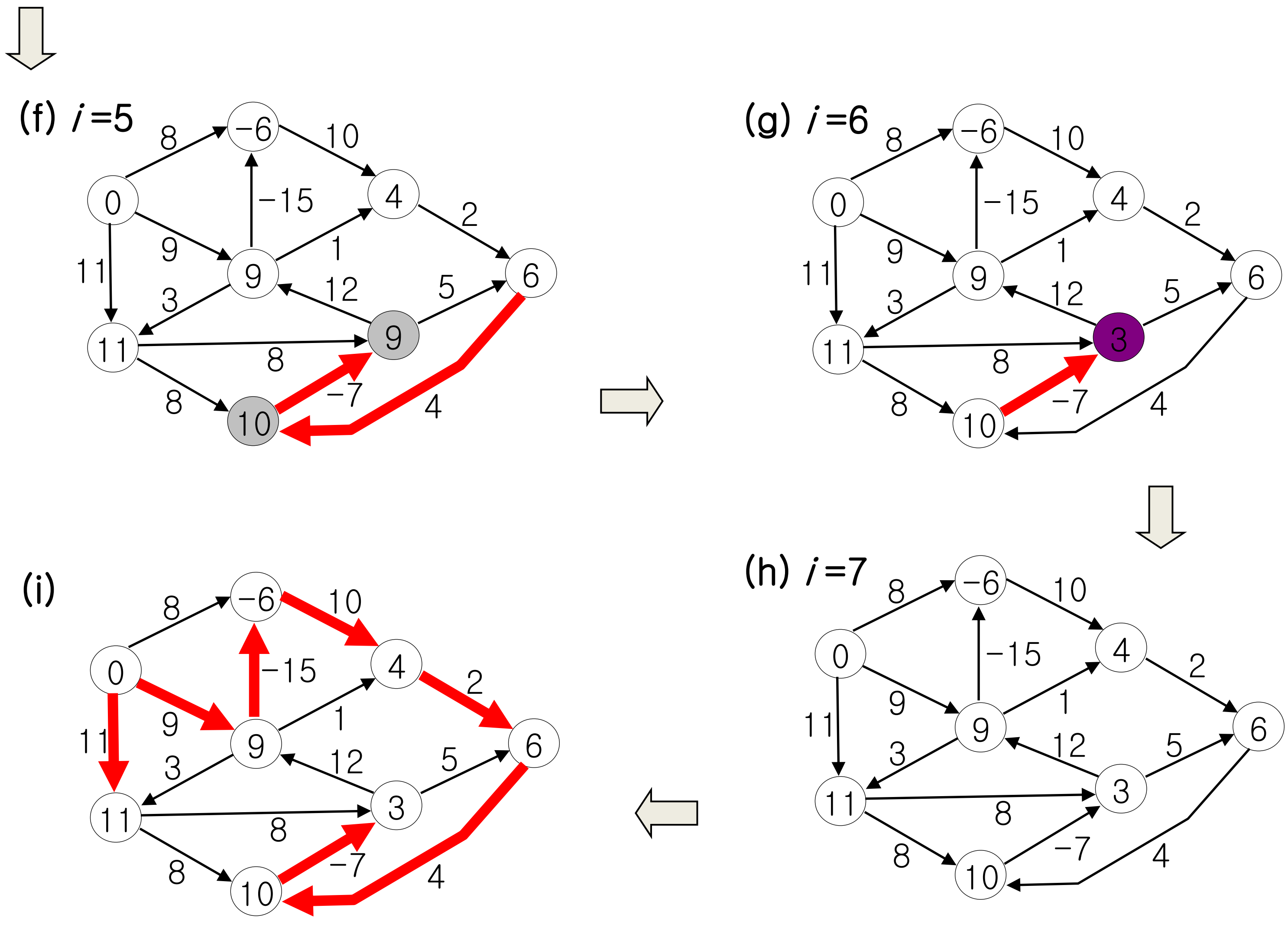
BellmanFord(G, r)
{
    for each $u \in V$
        $d_u \leftarrow \infty$;
    d[r] $\leftarrow$ 0;
    for $i \leftarrow 1$ **to** $|V|$-1
        for each $(u, v) \in E$
            if (d[u] + w[u, v] < d[v]$_v$) **then** {
                d[v] $\leftarrow$ d[u] + w[u, v] ;
                prev[v] $\leftarrow u$;
            }
    ▷ **음의 싸이클 존재 여부 확인**
    for each $(u, v) \in E$
        if (d[u] + w[u, v] < d[v]$_v$) **output** "**해없음**";
}

이완(relaxation)

✓수행 시간: $\Theta(|E||V|)$

벨만-포드 알고리즘의 작동 예





(f) i =5
0
−6
4
9
6
11
9
10
8
10
−15
2
9
1
11
3
12
5
8
8
−7
4
(g) i =6
0
−6
4
9
6
11
3
10
8
10
−15
2
9
1
11
3
12
5
8
8
−7
4
(h) i =7
0
−6
4
9
6
11
3
10
8
10
−15
2
9
1
11
3
12
5
8
8
−7
4
(i)
0
−6
4
9
6
11
3
10
8
10
−15
2
9
1
11
3
12
5
8
8
−7
4

동적 프로그래밍으로 본 벨만-포드 알고리즘

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

✓ 재귀적 관계

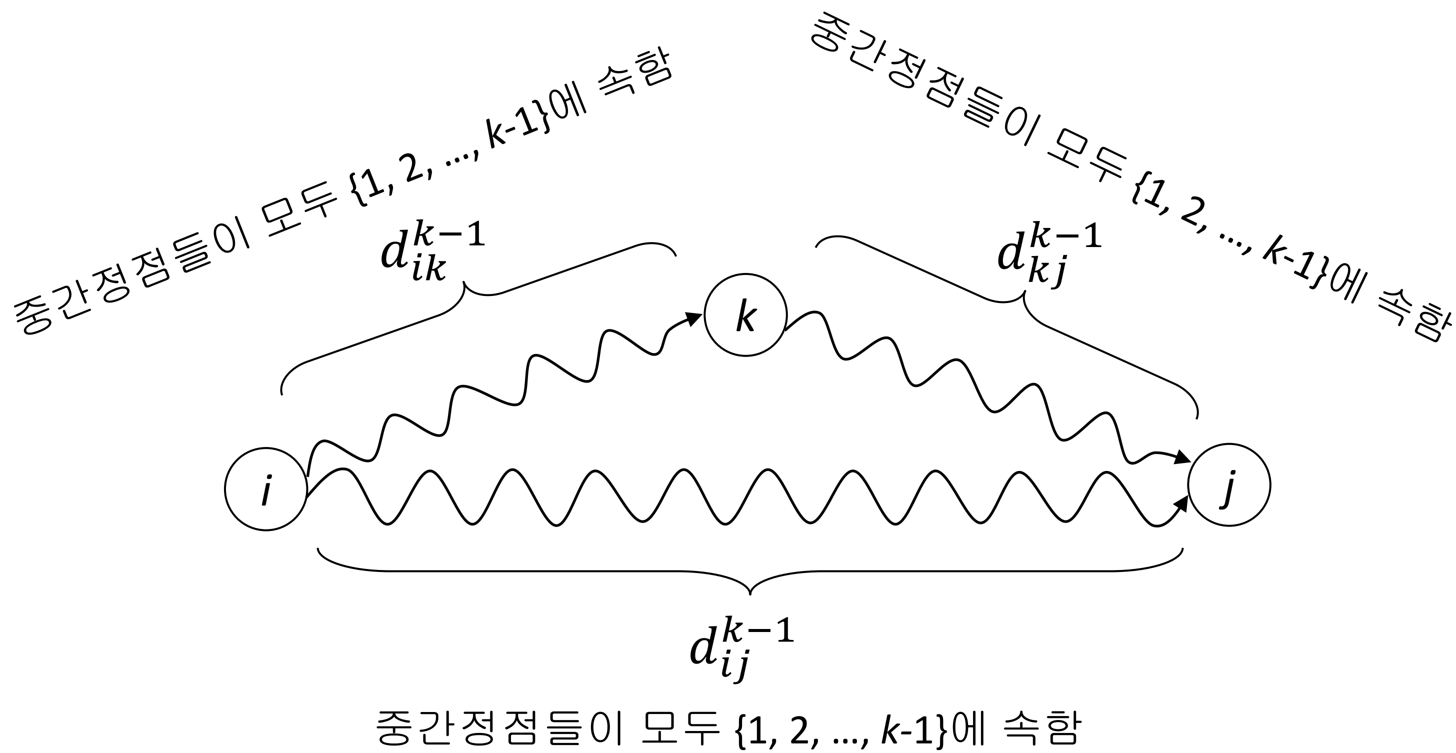
$$\begin{cases} d_v^k = \min\limits_{\text{for 모든 간선 } (u,\, v)} \{d_u^{k-1} + w_{uv}\}, & k > 0 \\ d_r^0 = 0 \\ d_t^0 = \infty, & t \neq r \end{cases}$$

플로이드-워샬Floyd-Warshall 알고리즘

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

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

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



플로이드-워샬 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 {dᵏ⁻¹ij , dᵏ⁻¹ik + dᵏ⁻¹kj};
}
```

$$d^{0}_{ij} \leftarrow w_{ij}; \qquad d^{k}_{ij} \leftarrow \min\{d^{k-1}_{ij},\ d^{k-1}_{ik} + d^{k-1}_{kj}\};$$

✓수행시간: $\Theta(|V|^3)$

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