『프림, 크루스칼, 다익스트라 참고 자료』

* 모든 페이지는 다른 전공 강의의 과거 강의 자료로부터 가져온 것이므로 배포, 타인에게 유출 금지

2~10 페이지 : 이산수학 강의자료. MST, 프림, 크루스칼 알고리즘에 대한 내용.
교재는『4차 산업혁명 시대의 이산수학』, 김대수 지음, 생능출판

11~12 페이지 : 컴퓨터네트워크 강의자료. 다익스트라 알고리즘(필기 없음)

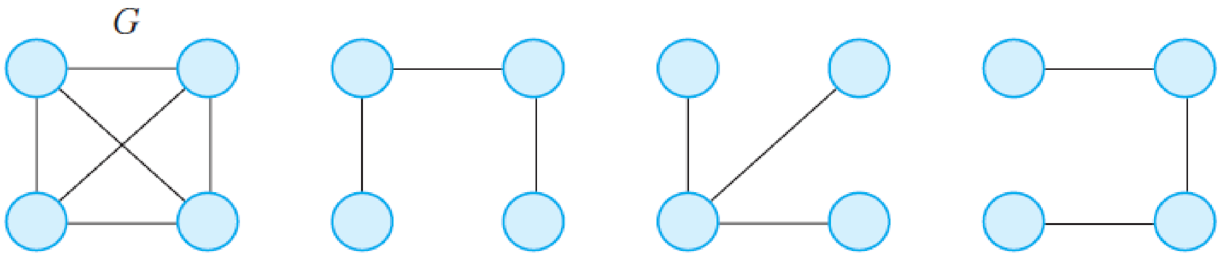
13~14 페이지 : 컴퓨터네트워크 강의자료. 다익스트라 알고리즘(필기 있음)
강의 교재 있긴 한데 전혀 도움 안 됨(설명 없음).

생성 트리 (Spanning Tree)

- **Spanning tree: 그래프 G의 모든 노드를 포함(연결)하는 트리 (subgraph)**

사이클 X

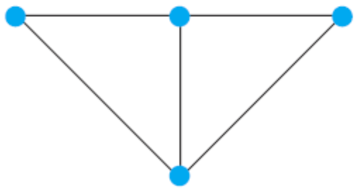
주어진 그래프 G와 그것의 3가지 생성 트리들



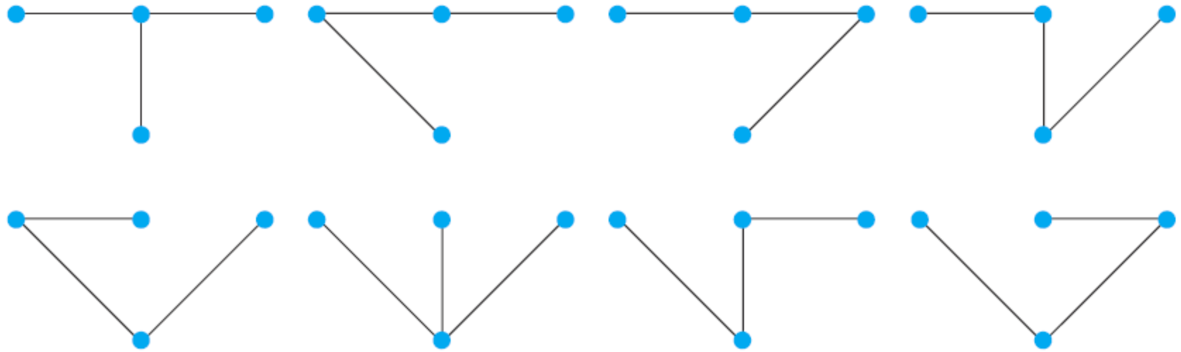
〈그림 8.13〉 주어진 그래프와 그것의 생성 트리들

생성 트리의 예

예제 8-5 다음과 같은 그래프 G의 생성 트리를 모두 구해보자.



풀이 G의 생성 트리는 모두 8가지인데 다음과 같다.



최소비용 생성 트리 (Minimum Spanning Tree, MST)

- Weighted graph에서 모든 vertex를 연결하는 spanning tree 중에서 edge의 weight 총합이 minimum인 것
  - 가장 싼 비용으로 6 개의 도시를 모두 연결할 수 있는 통신 망 구축하기
- MST 구하는 알고리즘
  - 프림(Prim)의 알고리즘 (하나의 node 에서 시작)
  - 크루스칼(Kruskal)의 알고리즘 (가장 weight가 작은 edge에서 시작)

Prim's Algorithm

알고리즘 1	프림의 알고리즘(Prim's algorithm)

주어진 가중 그래프 $G = (V, E)$에서 $V = \{1, 2, \cdots, n\}$이라고 하자.

(1) 노드의 집합 U를 1로 시작한다. u ← 1

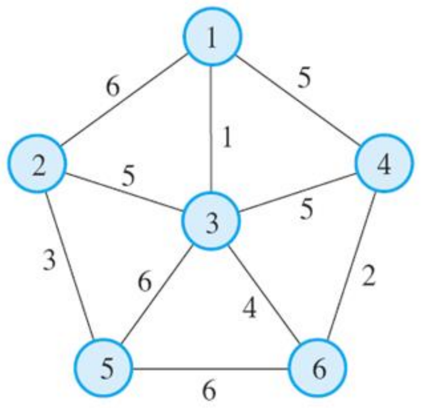
(2) $u \in U,\ v \in V - U$일 때 U와 $V - U$를 연결하는 가장 짧은 연결선인 (u, v)를 찾아서 v를 U에 포함시킨다. 이때 (u, v)는 사이클(cycle)을 형성하지 않는 것이라야 한다.

(3) (2)의 과정을 $U = V$ 때까지 반복한다.

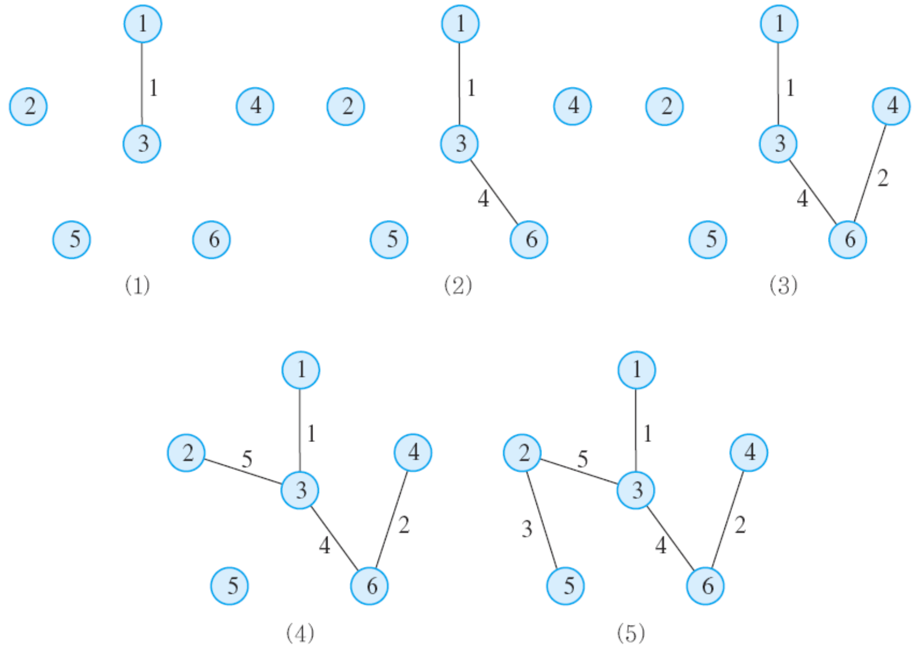
u : 이미 MST에 들어간 노드
v-u : 아직 MST에 안 들어간 노드

Prim 알고리즘 예

프림의 알고리즘으로 MST 구하는 과정



〈그림 8.14〉 가중 그래프



〈그림 8.15〉 프림의 알고리즘에 의한 MST의 생성 과정

Kruskal's Algorithm

알고리즘 2 크루스칼의 알고리즘(Kruskal's algorithm)

주어진 가중 그래프 $G = (V, E)$에서 $V = \{1, 2, \cdots, n\}$이라고 하고 T를 연결선의 집합이라고 하자.

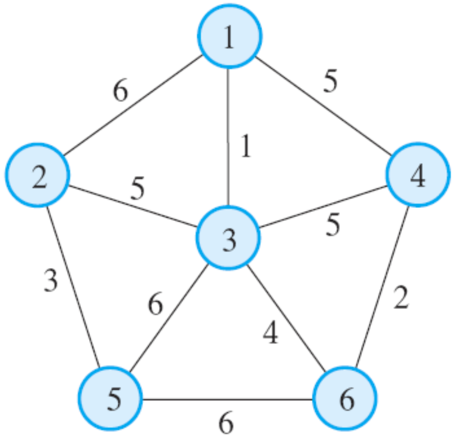
(1) T를 ϕ으로 놓는다.

(2) 연결선의 집합 E를 비용이 적은 순서로 정렬한다.

(3) 가장 최소값을 가진 연결선 (u, v)를 차례로 찾아서 (u, v)가 사이클을 이루지 않으면 (u, v)를 T에 포함시킨다.

(4) (3)의 과정을 $|T| = |V| - 1$일 때까지 반복한다.

Kruskal 알고리즘 예



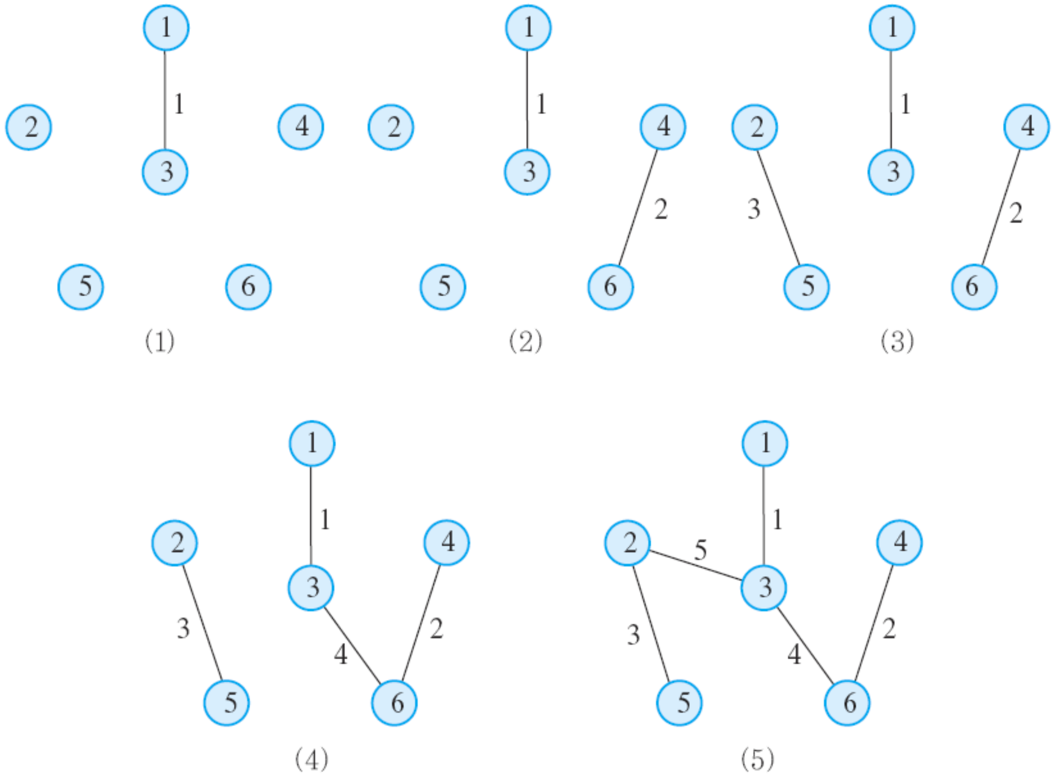
〈그림 8.14〉 가중 그래프

먼저 비용이 적은 순서로 나열

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

비용이 적은 순서대로 적용

단, 같은 값일 때는 어느 것을 선택하여
연결해도 관계없음

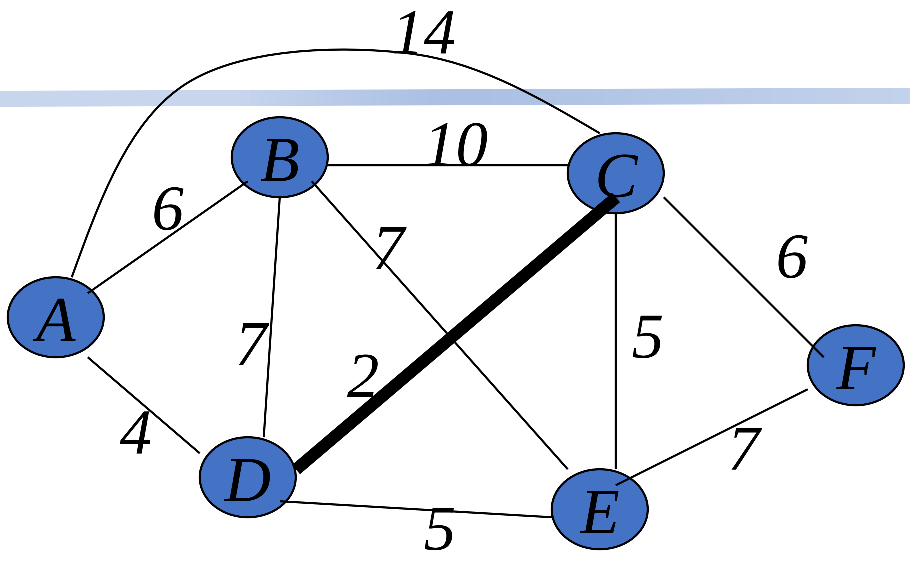


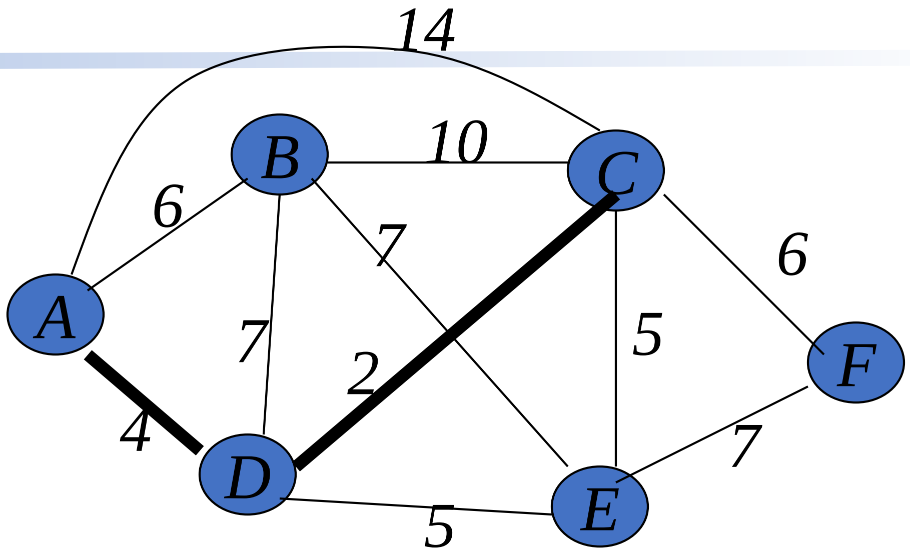
〈그림 8.16〉 크루스칼의 알고리즘에 의한 MST의 생성 과정

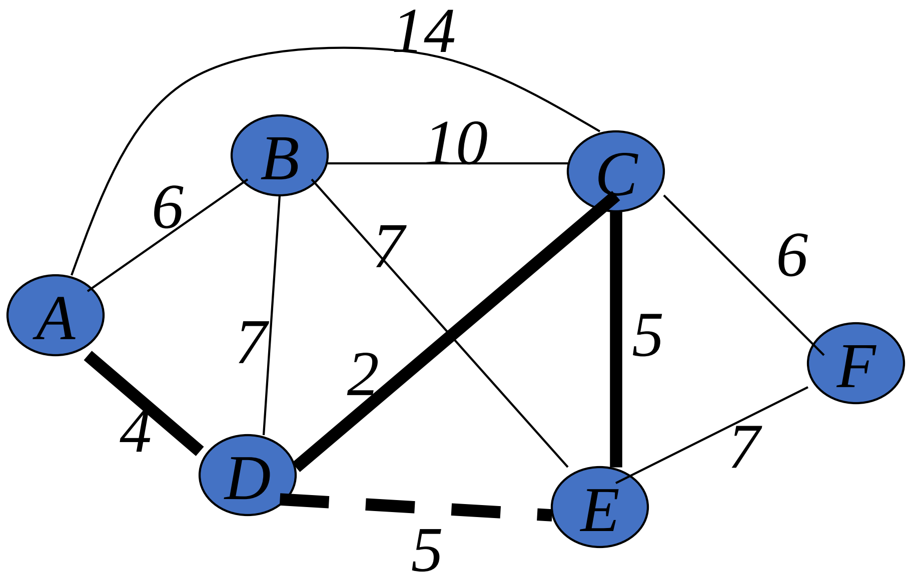
Kruskal's Algorithm

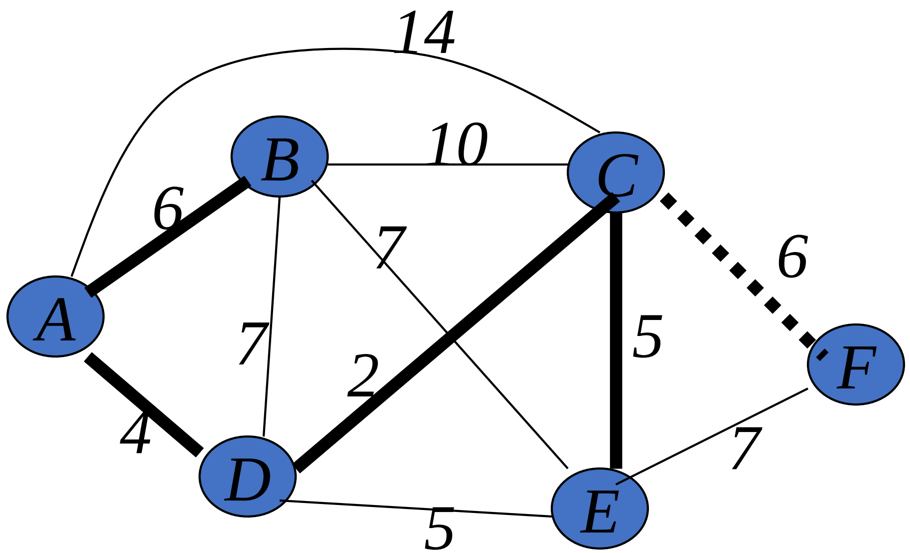
유사코드로 작성된 크루스칼 알고리즘

```
// Let G = <V, E>. V :set of vertices, E :set of edges
//                            N :the number of vertices in V
T = 0; // T is a set of edges (set to empty)
while (T contains less than N-1 edges && E not Empty)
{
  choose the lowest weighted edge  e  from E;
  delete e from E;
  if e does not create a cycle in T, add e to T;
}
if T contains fewer than N -1 edges, there is no spanning tree
```


14
10
6
7
7
2
5
6
4
7
5
A
B
C
D
E
F


14
10
6
7
7
2
5
6
4
7
5
A
B
C
D
E
F

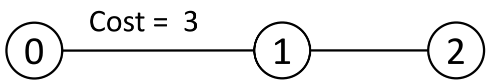

14
10
6
7
7
2
5
6
4
7
5
A
B
C
D
E
F


14
10
6
7
7
2
5
6
4
7
5
A
B
C
D
E
F

링크 상태 (Link State) 알고리즘
- Dijkstra 알고리즘

- 하나의 노드 s에서 모든 노드까지 최단 경로를 찾은 알고리즘
  - D = [ ] : 다른 점까지 거리를 나타내는 1차원 배열
    - 예) D[s] = 0 :자신까지의 거리는 0
  - V = { } : 방문한 노드 집합

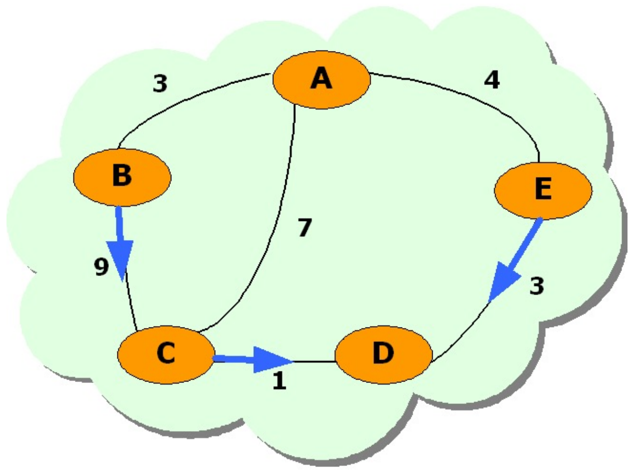
1. D, V의 초기화:
   - D는 s에 직접 인접한 노드의 값을 각 링크의 cost로 대입. 직접 인접하지 않으면 무한대
     - 예) D = [0, 3, ∞]
   - V = { s }



2. 미방문 노드 중에 가장 가까운 노드 u를 V에 추가
   u의 모든 이웃 w에 대해서
   D[w] = min(D[w], D[u] + cost[u][w])
3. 더 이상 미방문 노드를 고를 수 없을 때까지 2의 과정 반복

링크 상태 (Link State) 알고리즘

- Dijkstra 알고리즘



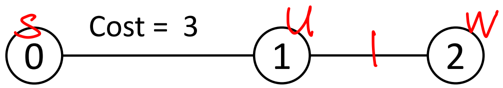
- A로부터의 최단경로를 찾는다면?
- 초기화
  - V = { A }
  - D = [0, 3, 7, ∞, 4] (A B C D E)
- 첫번째
  - V = { A, B }
  - B의 이웃은 A와 C
  - D[C] = min (D[C], D[B] + cost[B][C]) = min (7, 3 + 9) = 7
  - D = [0, 3, 7, ∞, 4]
- 두번째
  - V = { A, B, E }
  - E의 이웃은 A와 D
  - D[D] = min (D[D], D[E] + cost[E][D]) = min (∞, 4 + 3) = 7
  - D = [0, 3, 7, 7, 4]
- 세번째
  - V = { A, B, E, C }
  - C의 이웃은 B, D
  - D[B], D[D]는 더 작아질 수 없음
  - 네번째도 마찬가지
- 네번째를 마치고 모든 노드 방문 후 종료

링크 상태 (Link State) 알고리즘

- Dijkstra 알고리즘

- 하나의 노드 s에서 모든 노드까지 최단 경로를 찾은 알고리즘
  - D = [ ] : 다른 점까지 거리를 나타내는 1차원 배열
    - 예) D[s] = 0 :자신까지의 거리는 0
  - V = { } : 방문한 노드 집합

1. D, V의 초기화:
   - D는 s에 직접 인접한 노드의 값을 각 링크의 cost로 대입. 직접 인접하지 않으면 무한대
     - 예) D = [0, 3, ∞]

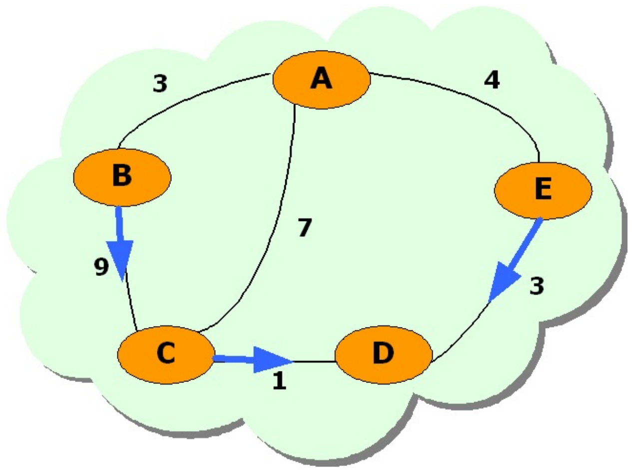


   - V = { s }
2. 미방문 노드 중에 가장 가까운 노드 u를 V에 추가
   u의 모든 이웃 w에 대해서
   D[w] = min(D[w], D[u] + cost[u][w]) → D(w)=4
3. 더 이상 미방문 노드를 고를 수 없을 때까지 2의 과정 반복

링크 상태 (Link State) 알고리즘

- Dijkstra 알고리즘

- A로부터의 최단경로를 찾는다면?
- 초기화
  - V = { A }
  - A B C D E
D = [0, 3, 7, ∞, 4]
- 첫번째
  - V = { A, B }
  - B의 이웃은 A와 C
  - 7 3 9
D[C] = min (D[C], D[B] + cost[B][C]) = min (7, 3 + 9) = 7
  - D = [0, 3, 7, ∞, 4]
- 두번째
  - V = { A, B, E }
  - E의 이웃은 A와 D
  - ∞ 4 3
D[D] = min (D[D], D[E] + cost[E][D]) = min (∞, 4 + 3) = 7
  - D = [0, 3, 7, 7, 4]



- 세번째
  - V = { A, B, E, C }
  - C의 이웃은 B, D
  - D[B], D[D]는 더 작아질 수 없음
  - 네번째도 마찬가지
- 네번째를 마치고 모든 노드 방문 후 종료