Chapter 4-1

Greedy Algorithm

그리디 (Greedy) 알고리즘

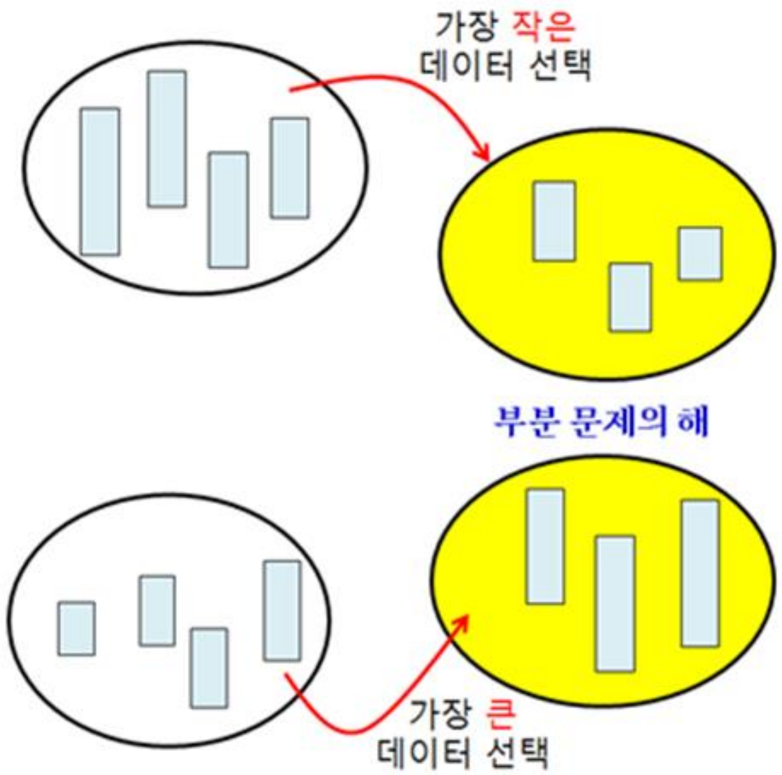
- Greedy 알고리즘은 최적화 문제를 해결하는 알고리즘
  - 최적화 (optimization) 문제
    - 가능한 해들 중에서 가장 좋은 (최대 or 최소) 해를 찾는 문제
- 욕심쟁이 방법, 탐욕적 방법, 탐욕 알고리즘 등으로 불림
- Greedy 알고리즘은 입력 데이터 간의 관계를 고려하지 않고 수행 과정에서 '욕심내어' 최소값 또는 최대값을 가진 데이터를 선택
  - 이러한 선택을 '근시안적'인 선택이라고 말하기도 함

그리디 (Greedy) 알고리즘

- Greedy 알고리즘은 근시안적인 선택으로 부분적인 최적해를 찾고, 이들을 모아서 문제의 최적해를 얻음

최솟값 찾는 문제

최댓값 찾는 문제



그리디 (Greedy) 알고리즘

- Greedy 알고리즘은 일단 한 번 선택하면 절대로 번복하지 않음
  - 즉, 선택한 데이터를 버리고 다른 것을 취하지 않음

- 이러한 특성 때문에 대부분의 Greedy 알고리즘들은 매우 단순하며, 또한 제한적인 문제만이 Greedy 알고리즘으로 해결됨

동전 거스름돈 문제

- 동전 거스름돈 (Coin Change) 문제를 해결하는 가장 간단하고 효율적인 방법
  - 남은 액수를 초과하지 않는 조건하에 탐욕적으로 가장 큰 액면의 동전을 취함
- 동전 거스름돈 문제의 최소 동전 수를 찾는 Greedy 알고리즘
  - 동전의 액면은 500원, 100원, 50원, 10원, 1원

동전 거스름돈 문제

- CoinChange()
  - 입력: 거스름돈 액수 W
  - 출력: 거스름돈 액수에 대한 최소 동전 수

1. change=W, n500=n100=n50=n10=n1=0 // n500, n100, n50, n10, n1은 각 동전의 개수
2. while (change >= 500) change = change-500, n500++
3. while (change >= 100) change = change-100, n100++
4. while (change >= 50) change = change-10, n50++
5. while (change >= 10) change = change-10, n10++
6. while (change >= 1) change = change-1, n1++,
7. Return (n500+n100+n50+n10+n1) // 총 동전 개수 리턴

CoinChange 알고리즘

- Greedy 알고리즘의 근시안적 특성
  - CoinChange 알고리즘은 남아있는 거스름돈인 change에 대해 가장 높은 액면의 동전을 거스르며,
  - 500원 동전을 처리하는 line 2에서는 100원, 50원, 10원, 1원 동전을 몇 개씩 거슬러 주어야 할 것인지에 대해서는 전혀 고려하지 않음

CoinChange 알고리즘

- 수행 과정
  - 760원의 거스름돈에 대해

CoinChange 알고리즘

- CoinChange 알고리즘의 문제점
  - 한국은행에서 160원 동전을 추가로 발행한다면, CoinChange 알고리즘이 항상 최소 동전 수를 계산할 수 있을까?
    - 거스름돈이 200원이라면, CoinChange 알고리즘은 160원 동전 1개와 10원 동전 4개로서 총 5개를 리턴

CoinChange 알고리즘의 결과

CoinChange 알고리즘

- CoinChange 알고리즘의 문제점
  - 200원에 대한 최소 동전 수는 100원짜리 동전 2개

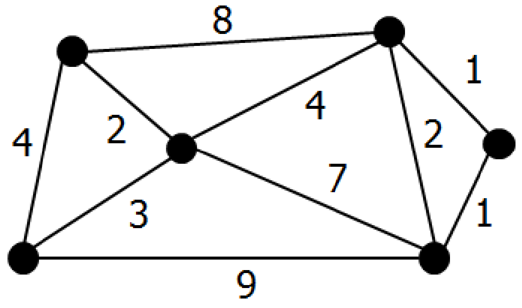
CoinChange 알고리즘의 결과

최소 동전의 거스름돈

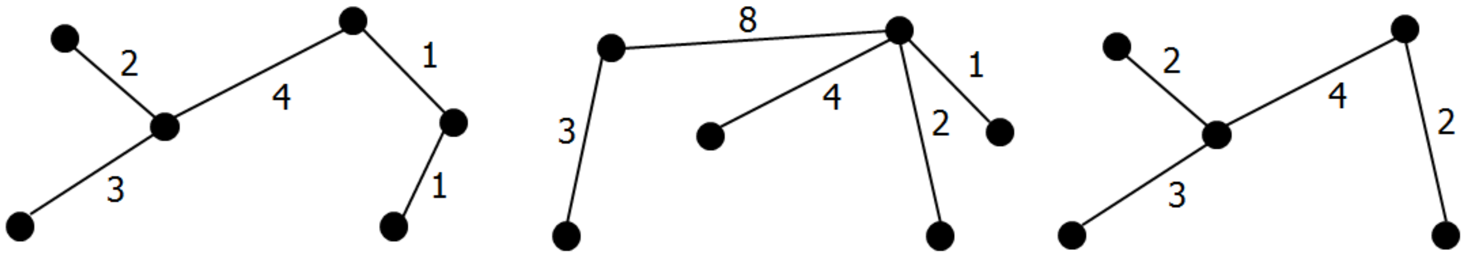
  - CoinChange 알고리즘은 항상 최적의 답을 주지는 못함
    - 그러나 실제로는 거스름돈에 대한 Greedy 알고리즘이 적용되도록 동전이 발행됨

최소 신장 트리 (Minimum Spanning Tree)

- 최소 신장 트리 정의
  - 주어진 가중치 그래프에서 사이클이 없이 모든 점들을 연결시킨 트리들 중 간선들의 가중치 합이 최소인 트리

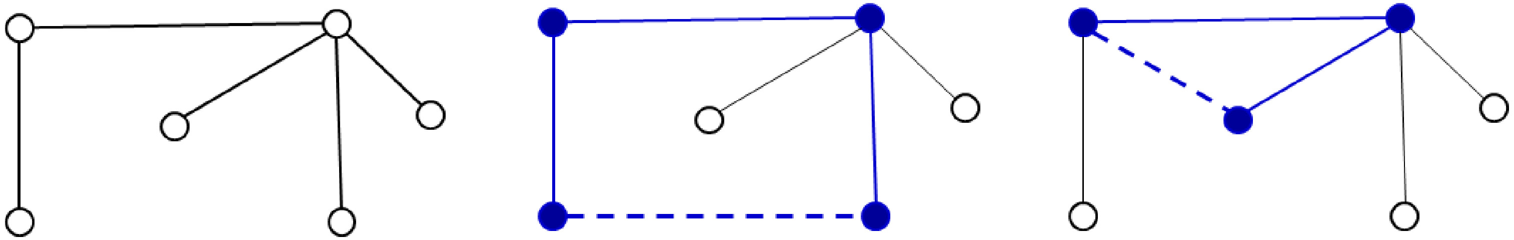


  - 다음 중 최소 신장 트리는?



최소 신장 트리 (Minimum Spanning Tree)

- 주어진 그래프의 신장 트리를 찾는 방법
  - 사이클이 없도록 모든 점을 연결
- 그래프의 점의 수 = n
  - 신장 트리에는 정확히 (n-1)개의 간선이 있음
  - 트리에 간선을 하나 추가시키면, 반드시 사이클이 만들어짐



트리 점선으로 된 선분을 추가하여 만들어진 사이클

최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- 크러스컬(Kruskal) 알고리즘
  - 가중치가 가장 작은 간선이 사이클을 만들지 않을 때에만 탐욕적으로 그 간선을 추가함
- 프림(Prim) 알고리즘
  - 임의의 점 하나를 선택한 후, (n-1)개의 간선을 하나씩 추가시켜 트리 생성
- 알고리즘의 입력은 1개의 연결 성분 (connected component)으로 된 가중치 그래프

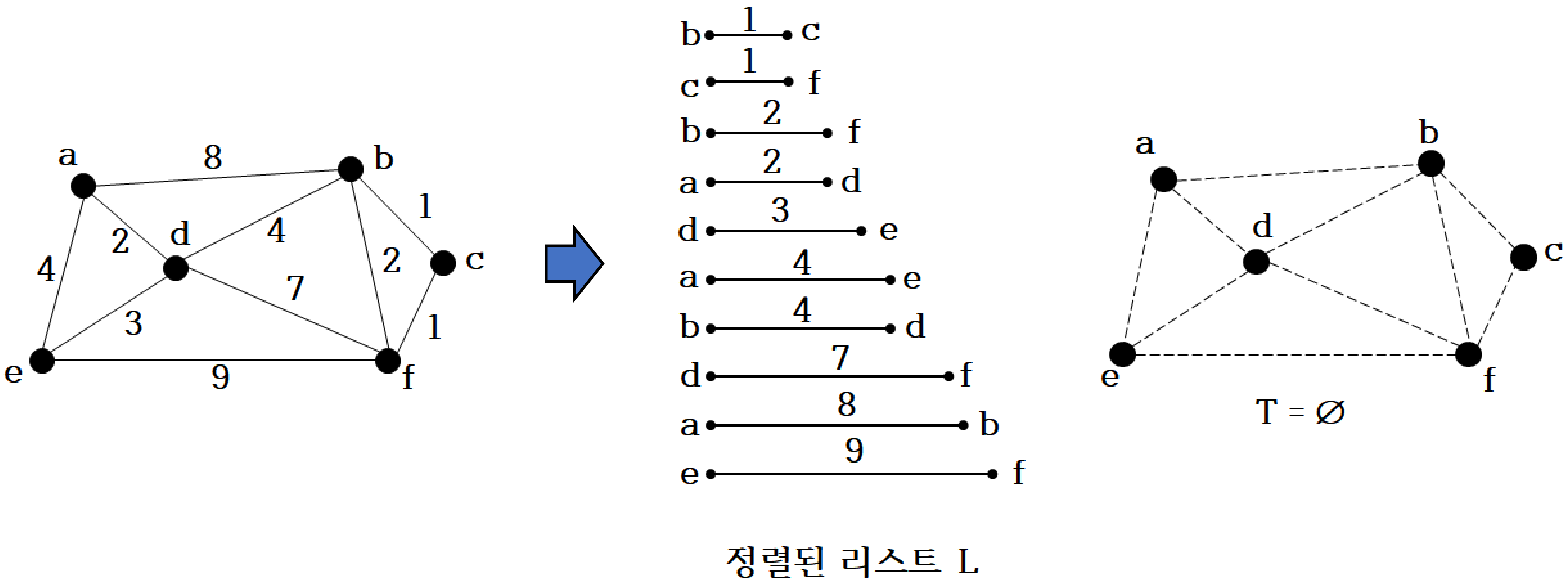
최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- KruskalMST(G)
  - 입력: 가중치 그래프 G=(V,E), |V|=n, |E|=m
  - 출력: 최소 신장 트리 T

```
가중치의 오름차순으로 간선들을 정렬: L = 정렬된 간선 리스트
T=Φ // 트리 T를 초기화
while (T의 간선 수 < n-1)
    L에서 가장 작은 가중치를 가진 간선 e를 가져오고, e를 L에서 제거
    if (간선 e가 T에 추가되어 사이클을 만들지 않으면)
        e를 T에 추가
    else        // e가 T에 추가되어 사이클이 생기는 경우
        e를 버림
Return 트리 T     // T는 최소 신장 트리
```

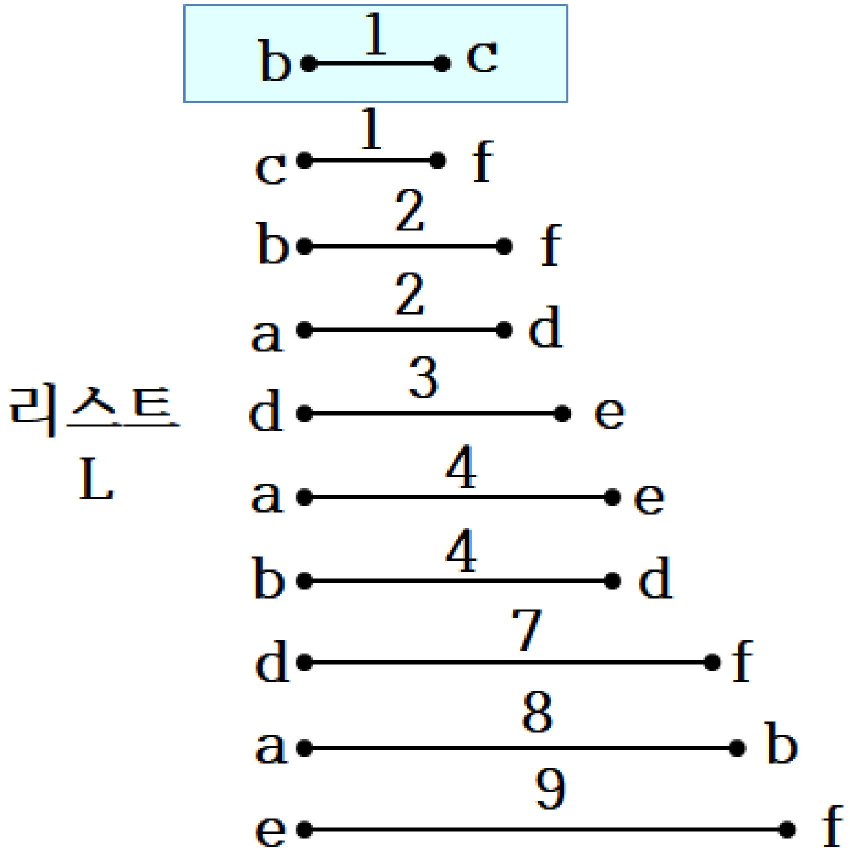
최소 신장 트리 (Minimum Spanning Tree) 알고리즘

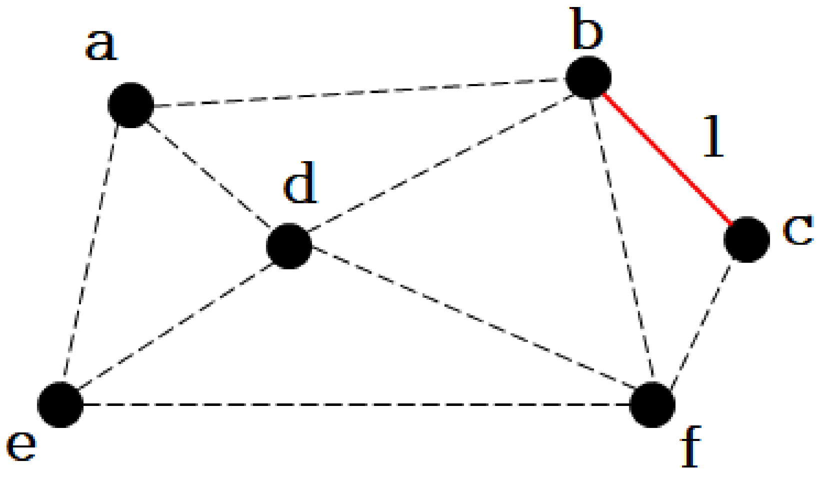
- Kruskal 알고리즘 수행 과정



최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- Kruskal 알고리즘 수행 과정

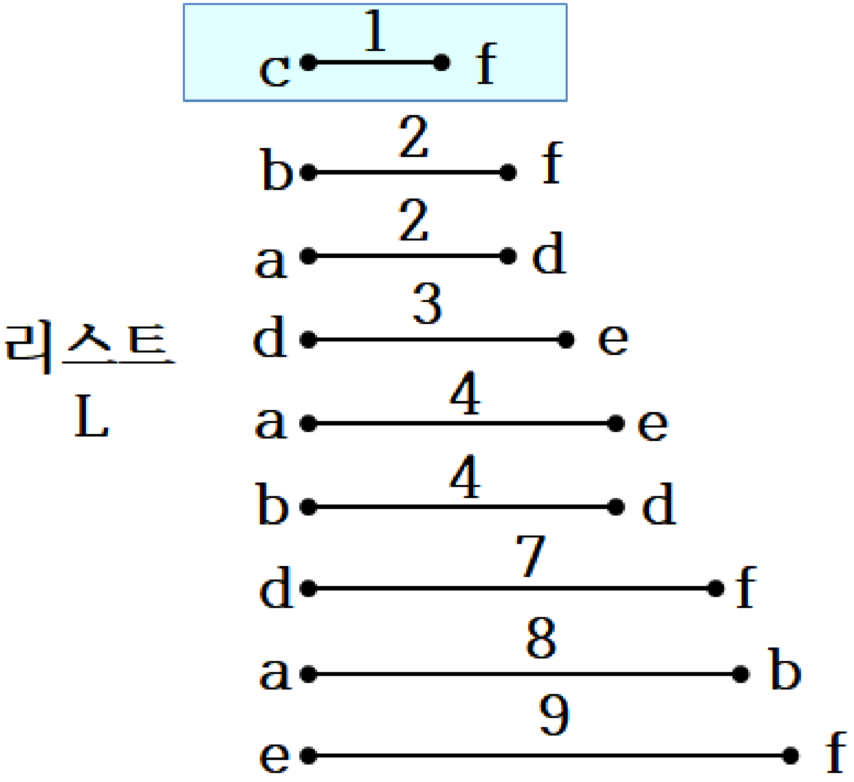


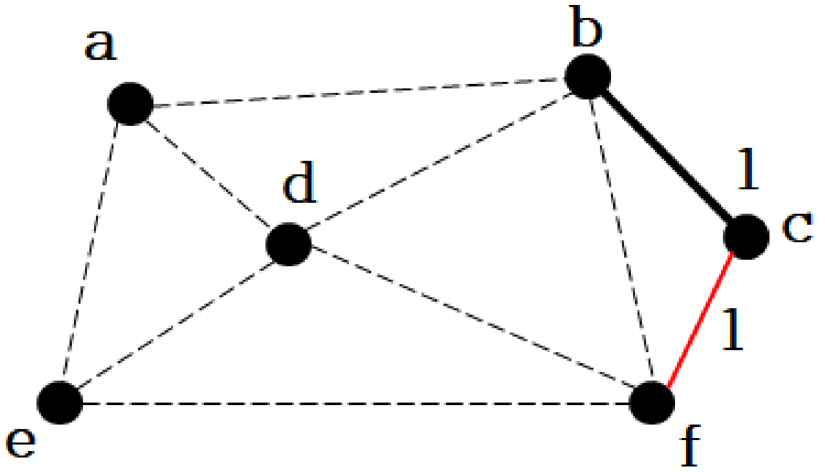


간선 (b, c) 추가

최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- Kruskal 알고리즘 수행 과정

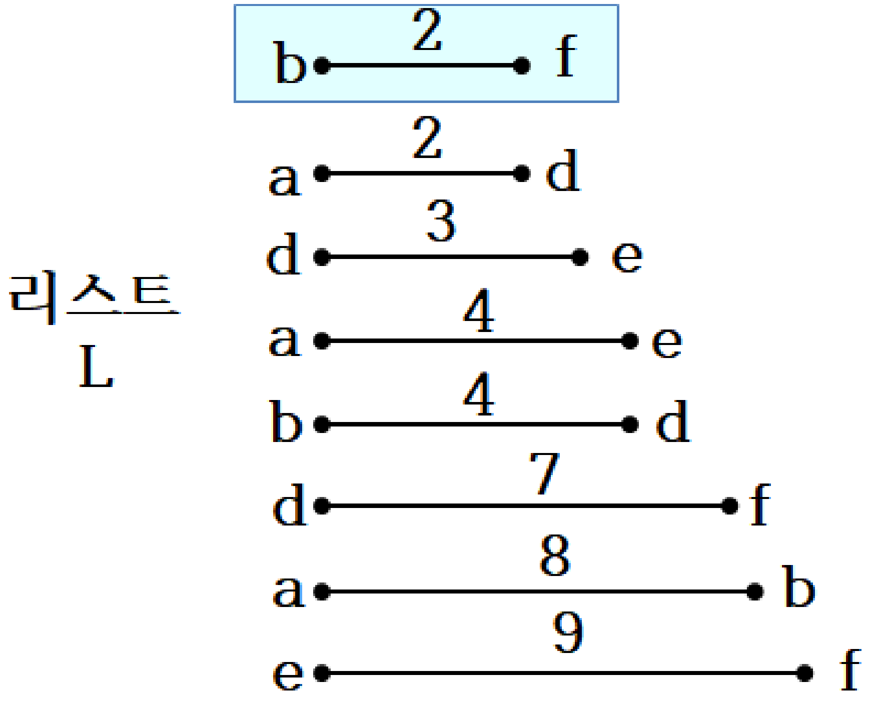


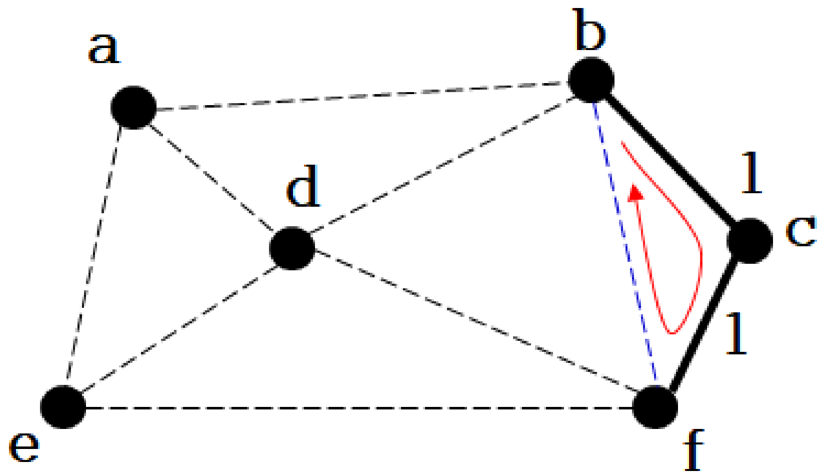


간선 (c, f) 추가

최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- Kruskal 알고리즘 수행 과정



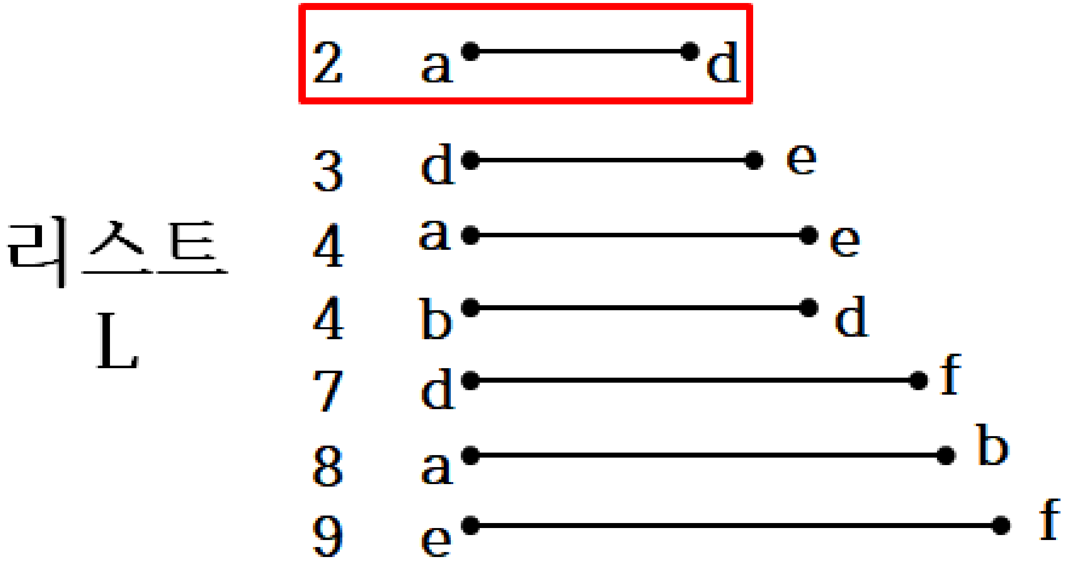


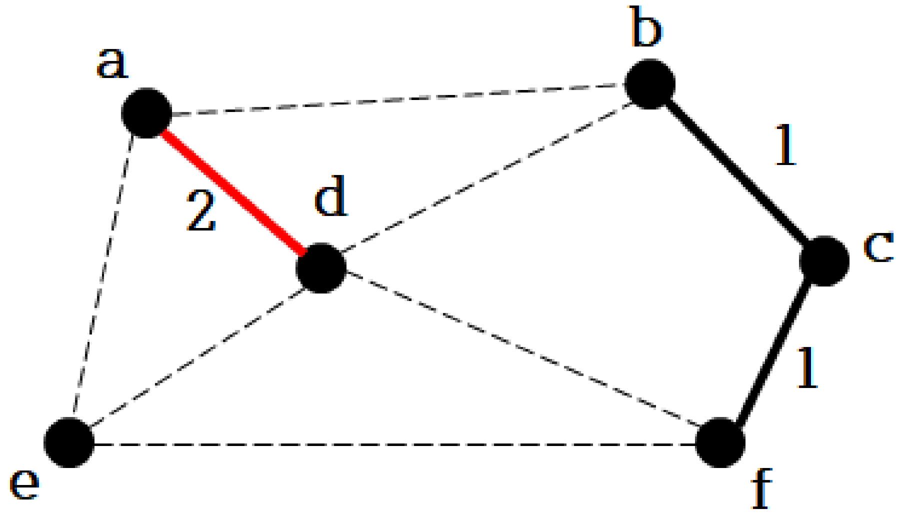
사이클 b-c-f-b

간선 (b, f) 버림

최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- Kruskal 알고리즘 수행 과정

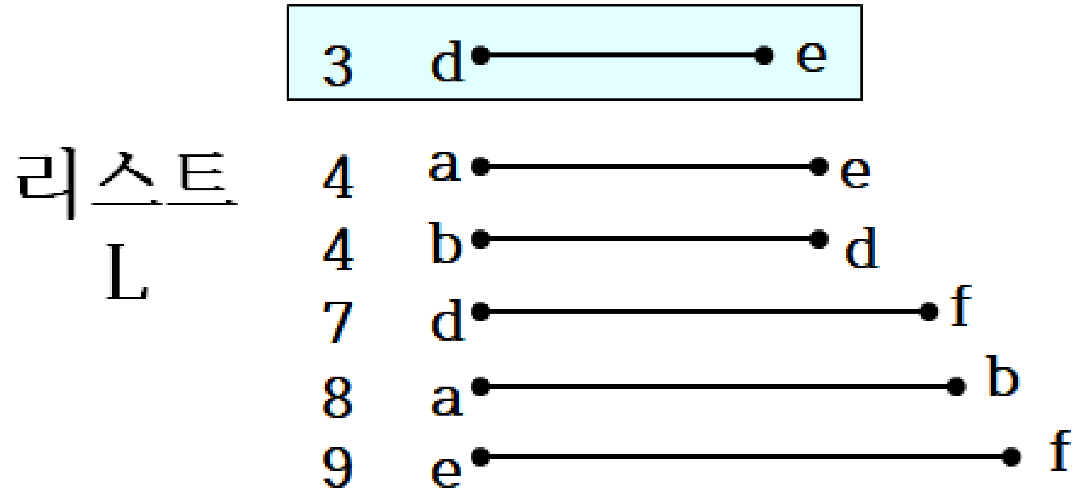


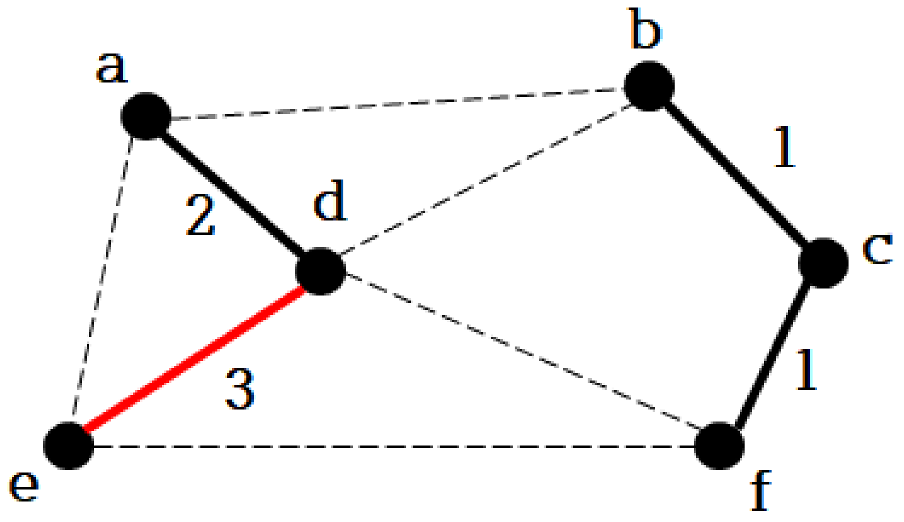


간선 (a, d) 추가

최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- Kruskal 알고리즘 수행 과정

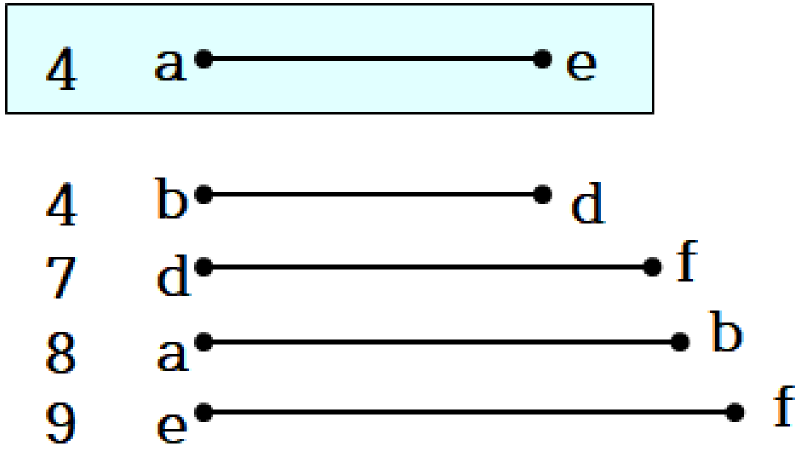


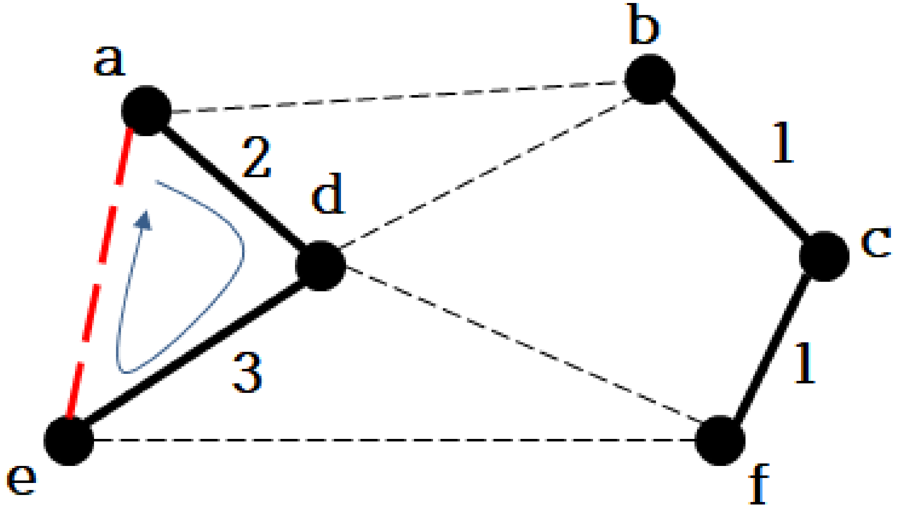


간선 (d, e) 추가

최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- Kruskal 알고리즘 수행 과정



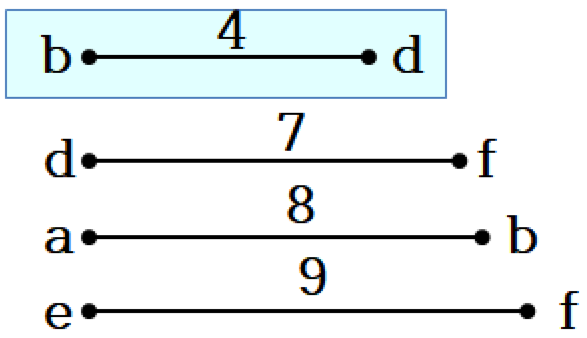


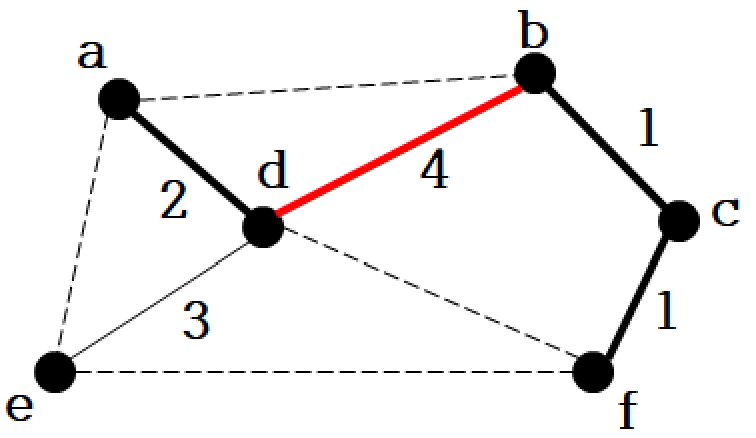
사이클a-d-e-a

최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- Kruskal 알고리즘 수행 과정

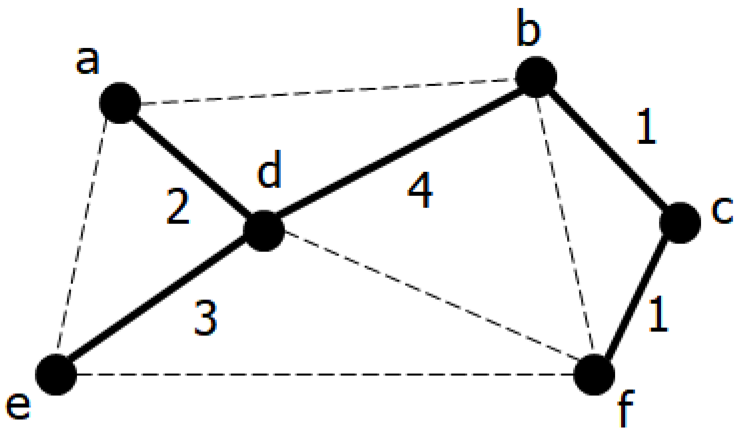
리스트 L





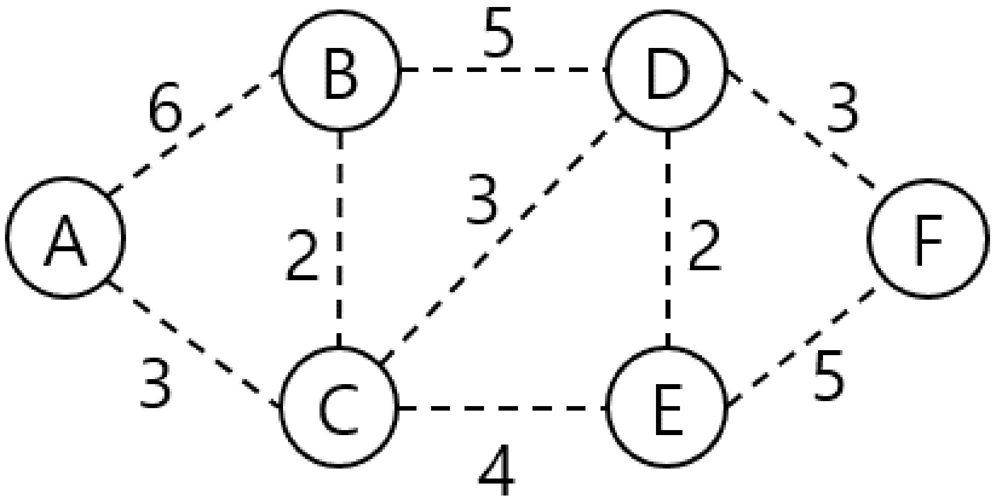
간선 (b, d) 추가

MST



최소 신장 트리 (Minimum Spanning Tree) 알고리즘

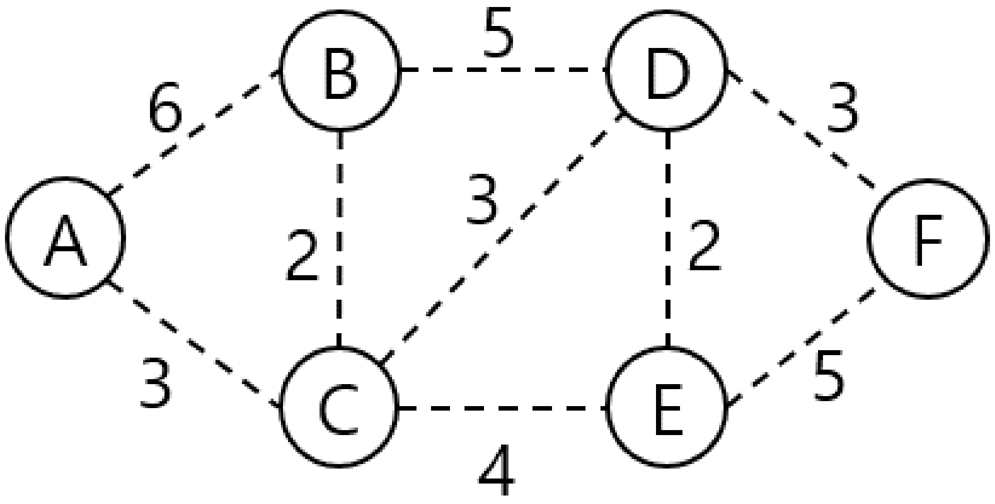
- Kruskal 알고리즘의 다른 예제



- a – c : 3
- a – b : 6
- b – c : 2
- b – d : 5
- c – e : 4
- c – d : 3
- d – e : 2
- d – f : 3
- e – f : 5

최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- Kruskal 알고리즘의 다른 예제

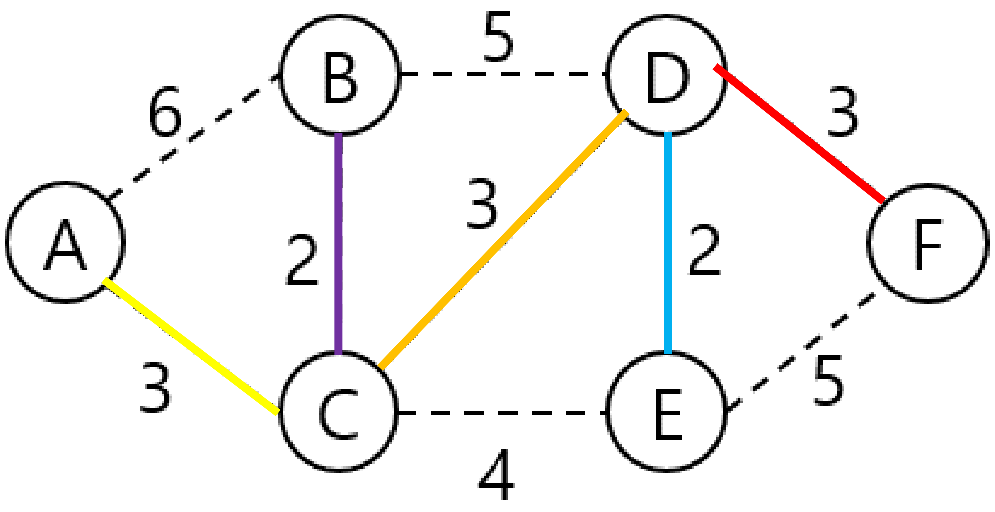


오름차순 정렬

- b – c : 2
- d – e : 2
- a – c : 3
- c – d : 3
- d – f : 3
- c – e : 4
- b – d : 5
- e – f : 5
- a – b : 6

최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- Kruskal 알고리즘의 다른 예제



오름차순 정렬

- b – c : 2
- d – e : 2
- a – c : 3
- c – d : 3
- d – f : 3
- c – e : 4
- b – d : 5
- e – f : 5
- a – b : 6

가중치의 합 = 13

최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- Kruskal 알고리즘 시간 복잡도
  - Line 1: 간선 정렬하는데 O(mlogm) 시간
    - m: 입력 그래프에 있는 간선의 수
  - Line 2: T를 초기화하는 것이므로 O(1) 시간
  - Line 3-8
    - while-loop는 최대 m번 수행
      - 그래프의 모든 간선이 while-loop 내에서 처리되는 경우
    - while-loop 내에서는 L로부터 가져온 간선 e가 사이클을 만드는지를 검사하는데 거의 O(1) 시간
  - Kruskal 알고리즘의 시간 복잡도: O(mlogm)

최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- 프림 (Prim)의 MST 알고리즘
  - 주어진 가중치 그래프에서 임의의 점 하나를 선택한 후, (n-1)개의 간선을 하나씩 추가시켜 트리 생성
  - 추가되는 간선은 현재까지 만들어진 트리에 연결시킬 때 탐욕적으로 항상 최소의 가중치로 연결되는 간선임

최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- 프림 (Prim)의 MST 알고리즘
  - PrimMST(G)
    - 입력: 가중치 그래프 G=(V,E), |V|=n, |E|=m
    - 출력: 최소 신장 트리 T

1. G에서 임의의 점 p를 시작점으로 선택 D[p] = 0 // D[v]는 T에 있는 u와 v를 연결하는 간선의 최소 가중치를 저장하기 위한 원소
2. for (점 p가 아닌 각 점 v에 대하여) { // 배열 D의 초기화
3. if (간선 (p, v)가 그래프에 있으면)
4. D[v] = 간선 (p, v)의 가중치
5. else
6. D[v] = ∞ }

최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- 프림 (Prim)의 MST 알고리즘

7. T = {p}   // 초기에 트리 T는 점 p만을 가짐

8. While (T에 있는 점의 수 < n) {

9. T에 속하지 않은 각 점 v에 대하여, D[v]가 최소인 점 v_{min}과 연결된 간선 (u, v_{min})을 T에 추가, 여기서 u는 T에 속한 점이고, 점 v_{min} 도 T에 추가

10. for (T에 속하지 않은 각 점 w에 대해서) {

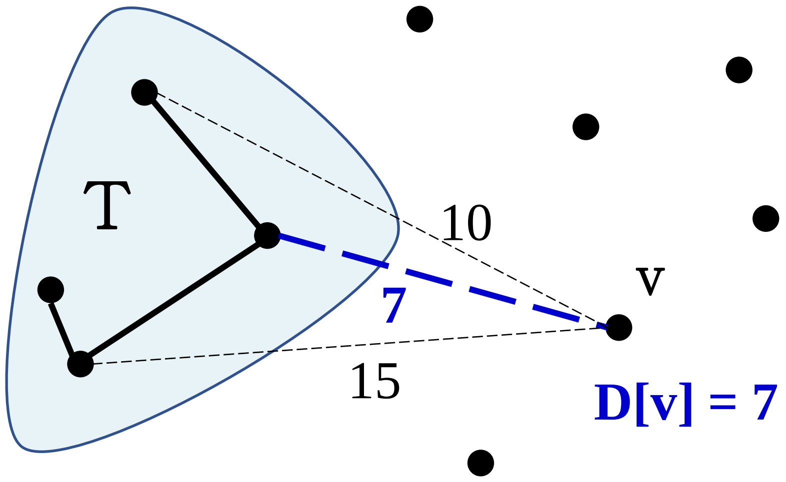
11. if (간선 (v_{min}, w)의 가중치 < D[w])

12. D[w] = 간선 (v_{min}, w)의 가중치   // D[w]를 갱신} }

13. return T   // T는 최소 신장 트리

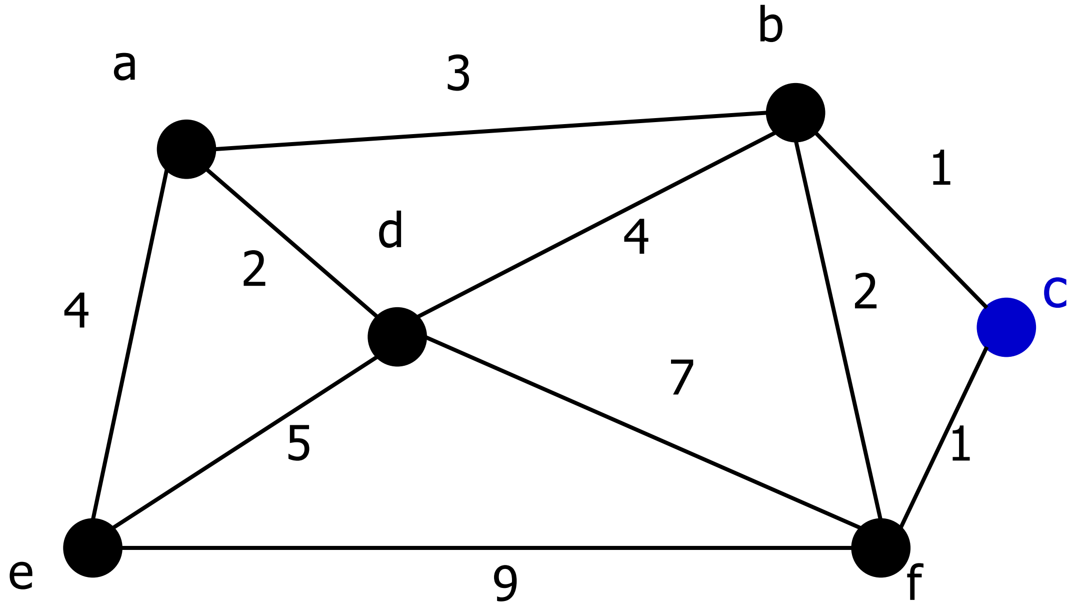
최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- 프림 (Prim)의 MST 알고리즘 수행과정
  - Line 1
    - 임의로 점 p를 선택하고, D[p] = 0으로 놓음
    - D[v]에는 점 v와 T에 속한 점들을 연결하는 간선들 중에서 최소 가중치를 가진 간선의 가중치를 저장
    - 그림에서 D[v]에는 10, 7, 15 중에서 최소 가중치인 7이 저장



최소 신장 트리 (Minimum Spanning Tree) 알고리즘

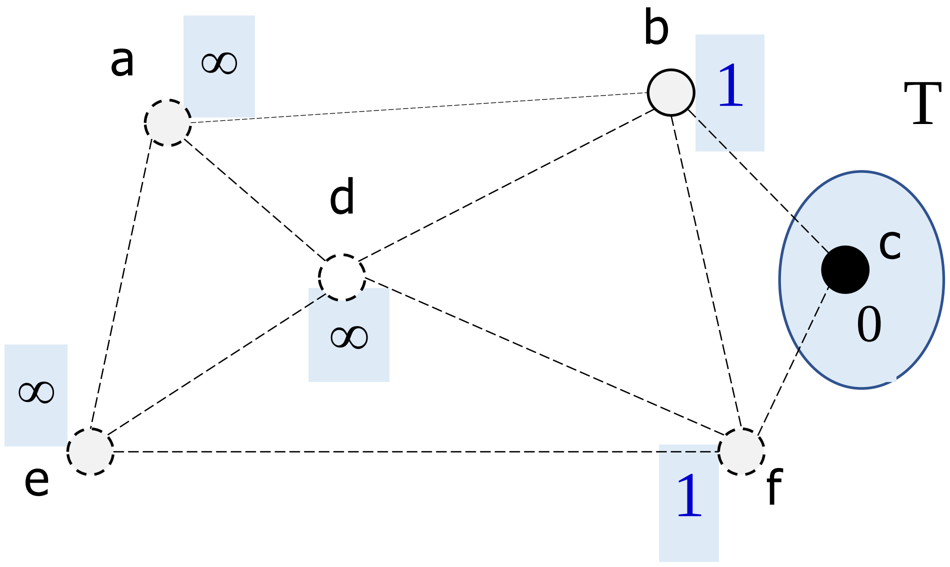
- 프림 (Prim)의 MST 알고리즘 수행과정



- Line 1: 임의의 점 c 선택, D[c] = 0으로 초기화

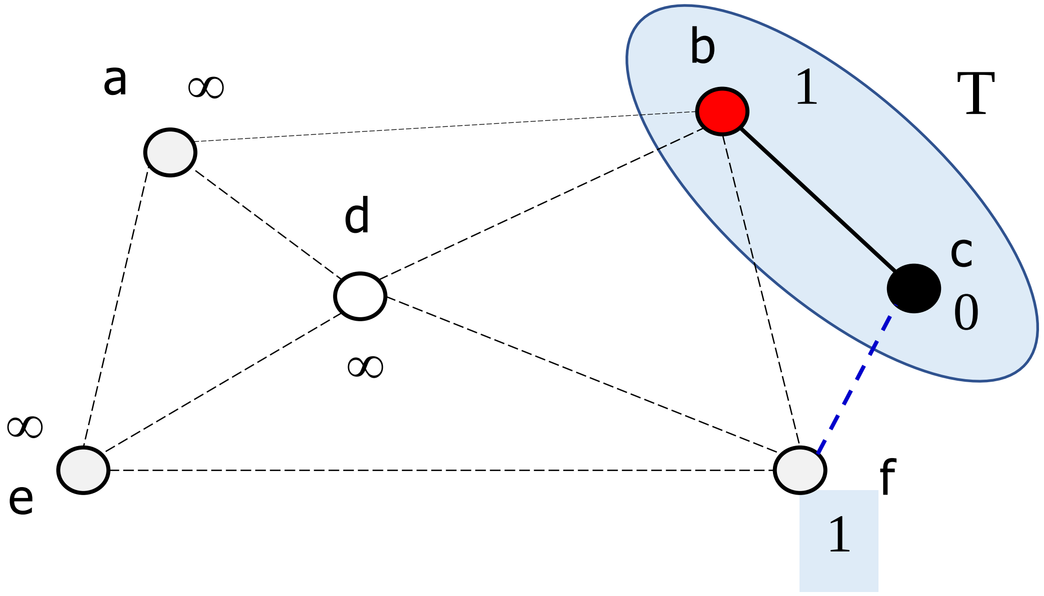
최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- 프림 (Prim)의 MST 알고리즘 수행과정
  - Line 2-6
    - 시작점 c와 간선으로 연결된 각 점 v에 대해서, D[v]를 각 간선의 가중치로 초기화
    - 점 c와 간선으로 연결되지 않은 나머지 각 점 v에 대해서, D[v]는 ∞로 초기화



최소 신장 트리 (Minimum Spanning Tree) 알고리즘

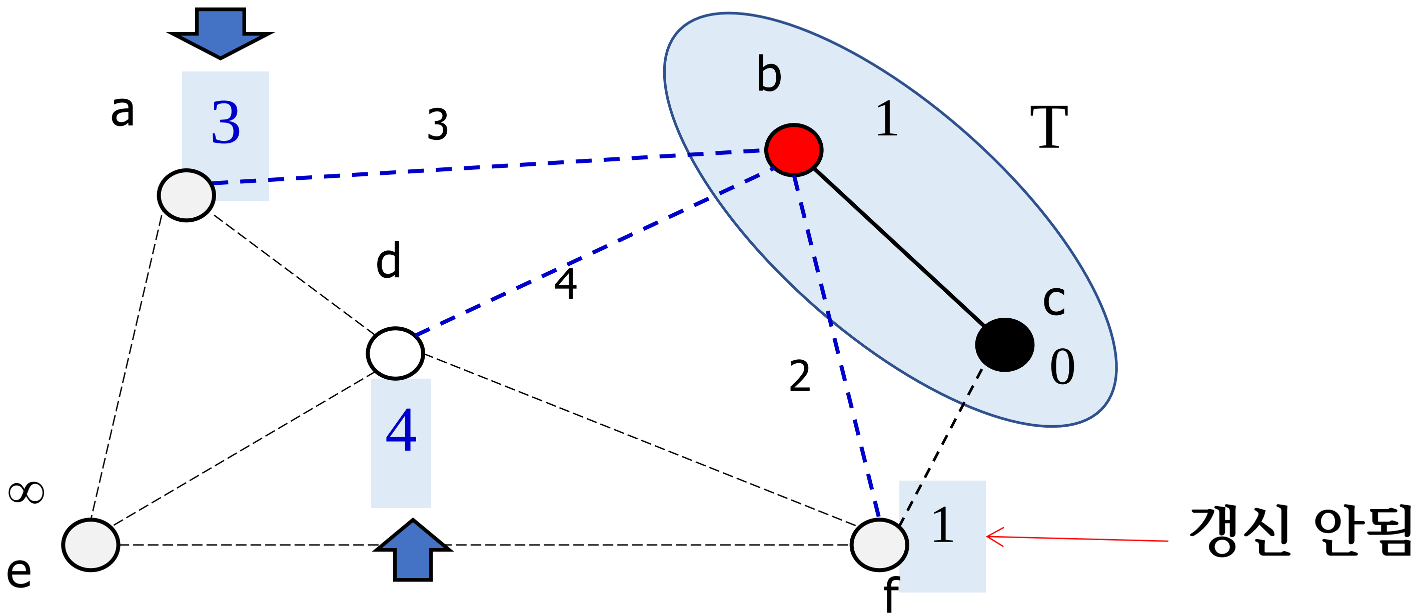
- 프림 (Prim)의 MST 알고리즘 수행과정
  - Line 7: T = {c}로 초기화
  - Line 8-9: T에 가장 가까운 점 b를 추가



$1 = \min\{\infty, 1, \infty, \infty, 1\}$

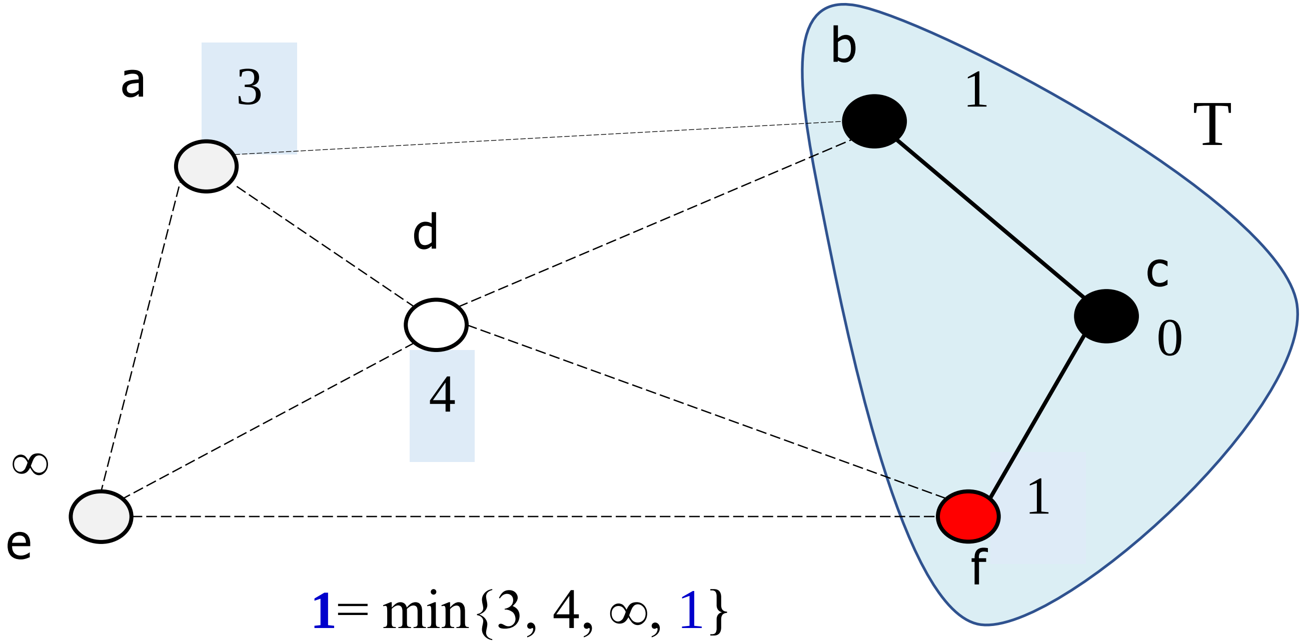
최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- 프림 (Prim)의 MST 알고리즘 수행과정
  - b에 연결된 a와 d의 D[a]와 D[d] 갱신



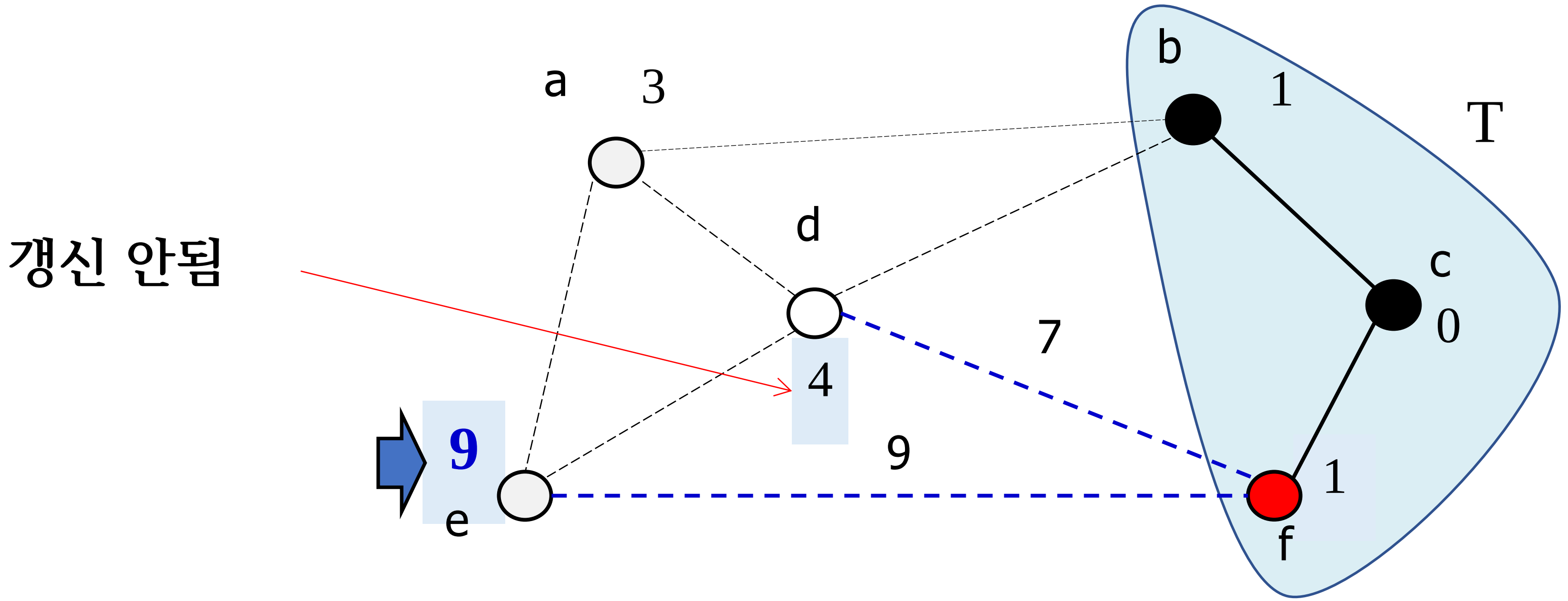
최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- 프림 (Prim)의 MST 알고리즘 수행과정
  - T에 가장 가까운 점 f를 추가



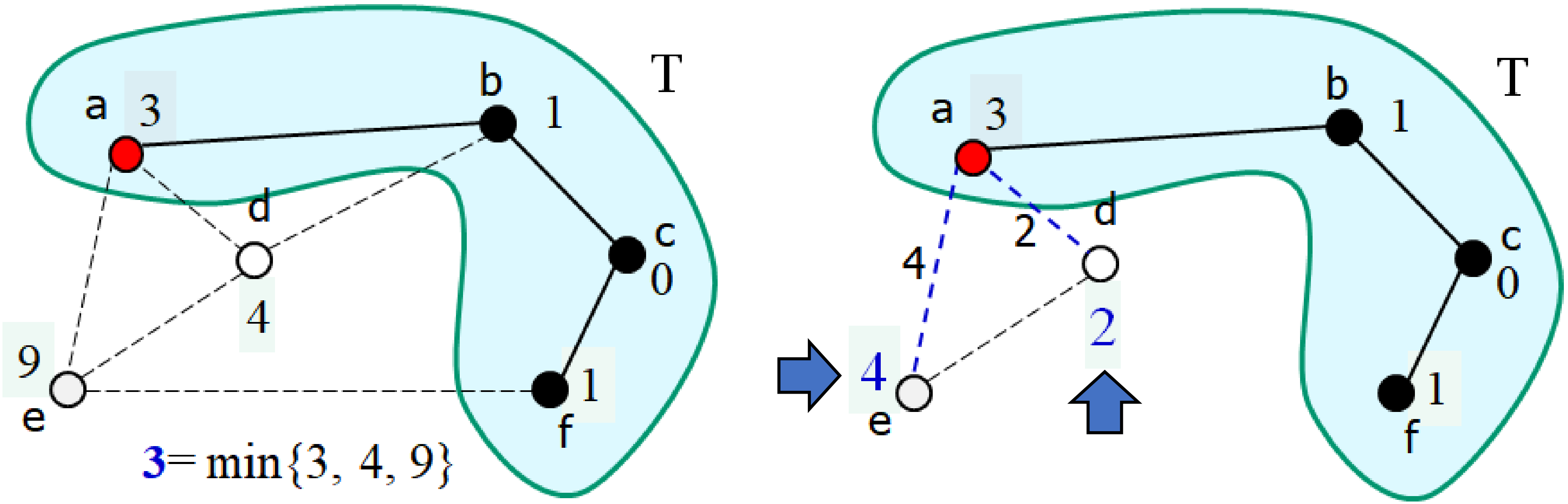
최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- 프림 (Prim)의 MST 알고리즘 수행과정
  - f에 연결된 e의 D[e] 갱신



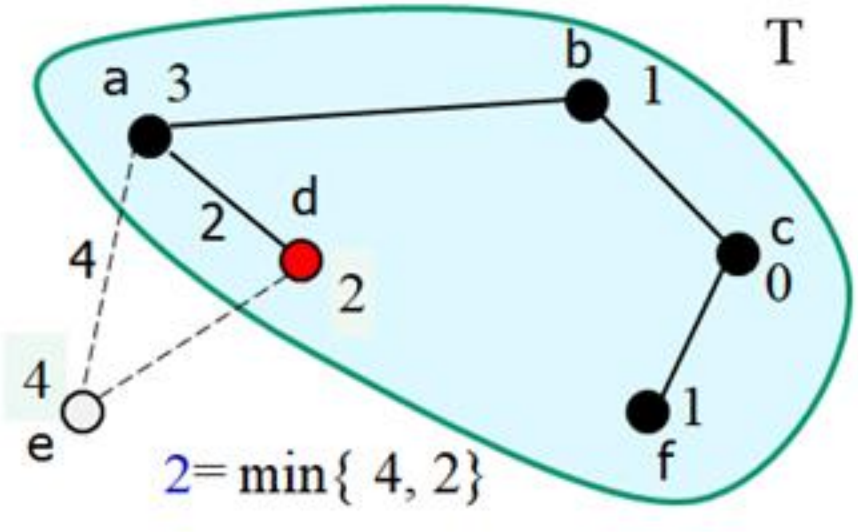
최소 신장 트리 (Minimum Spanning Tree) 알고리즘

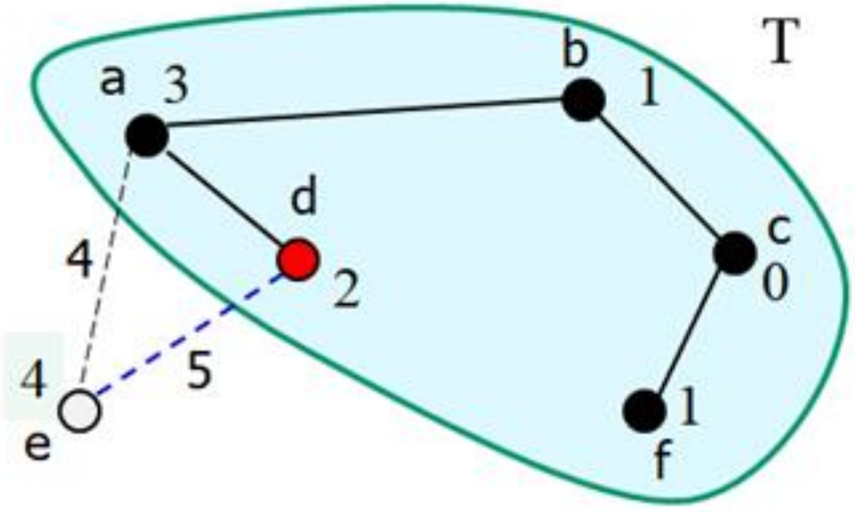
- 프림 (Prim)의 MST 알고리즘 수행과정
  - a를 T에 추가



최소 신장 트리 (Minimum Spanning Tree) 알고리즘

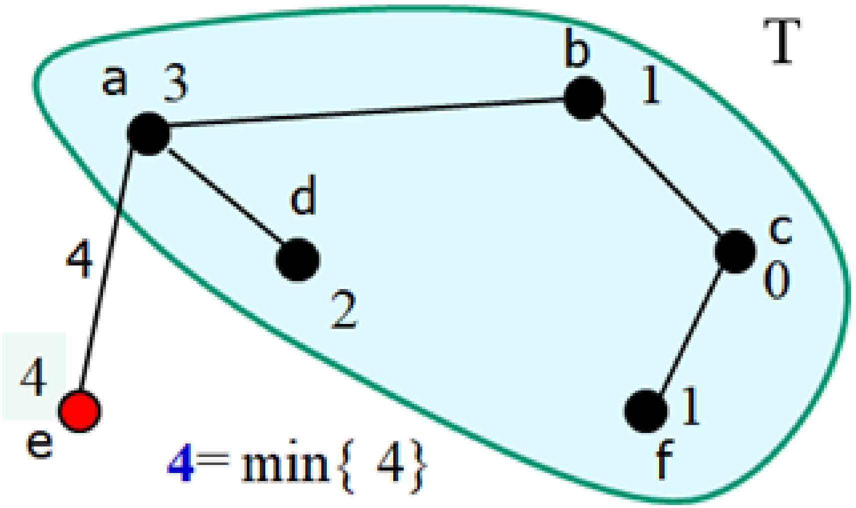
- 프림 (Prim)의 MST 알고리즘 수행과정
  - d를 T에 추가

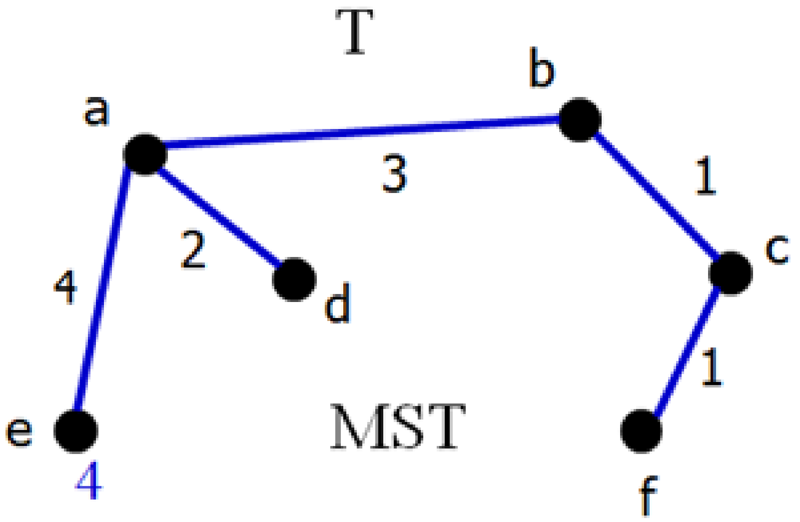




최소 신장 트리 (Minimum Spanning Tree) 알고리즘

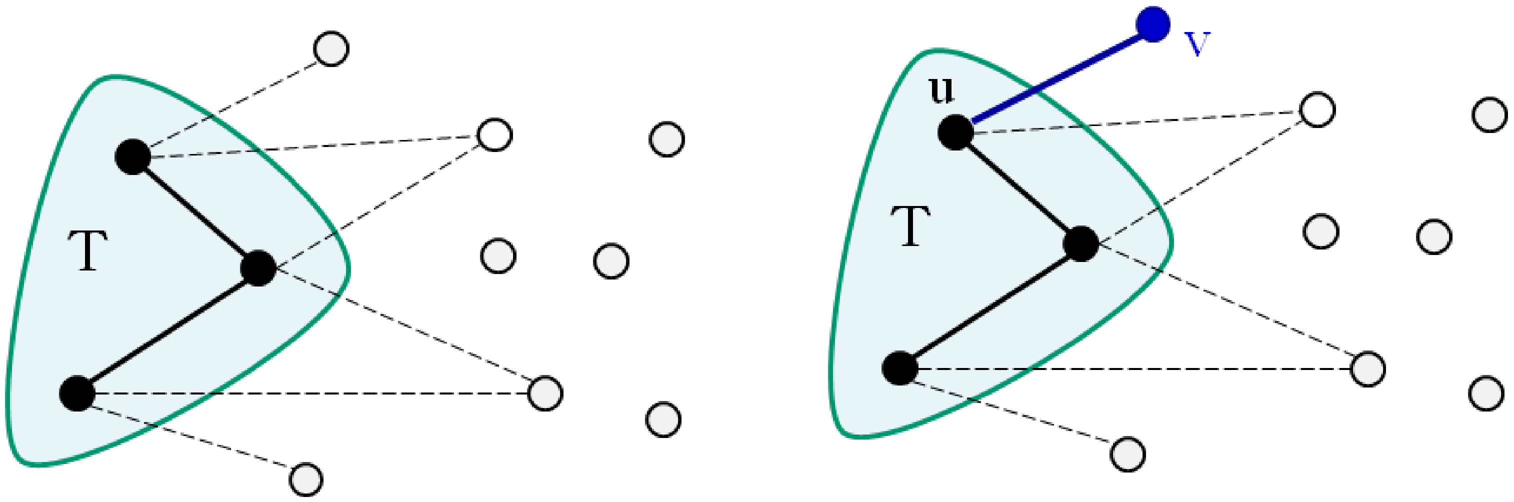
- 프림 (Prim)의 MST 알고리즘 수행과정
  - e를 T에 추가





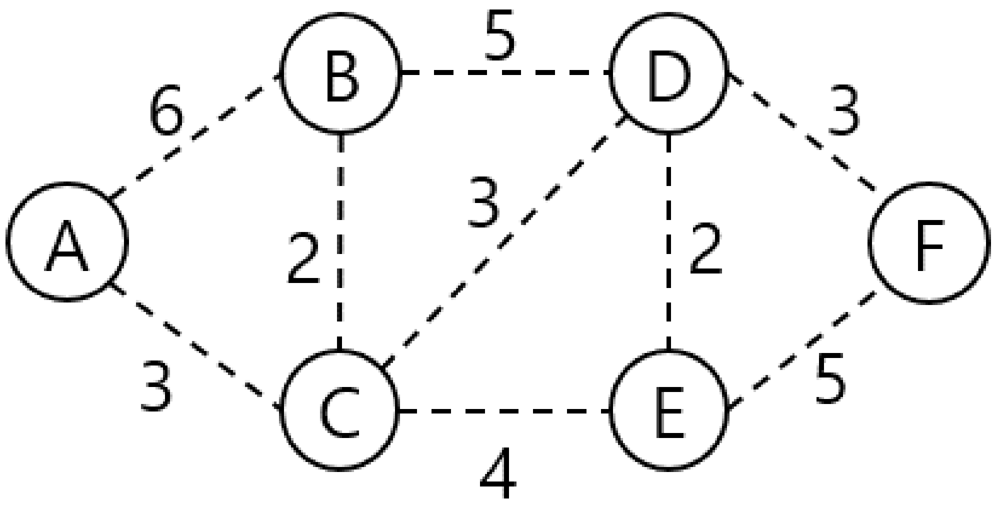
최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- 프림 (Prim)의 MST 알고리즘 수행과정
  - PrimMST가 찾은 T에는 사이클이 없는 이유
    - 프림 알고리즘은 T 밖에 있는 점을 항상 추가하므로 사이클이 만들어 지지 않음



최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- 프림 (Prim) MST 알고리즘의 다른 예

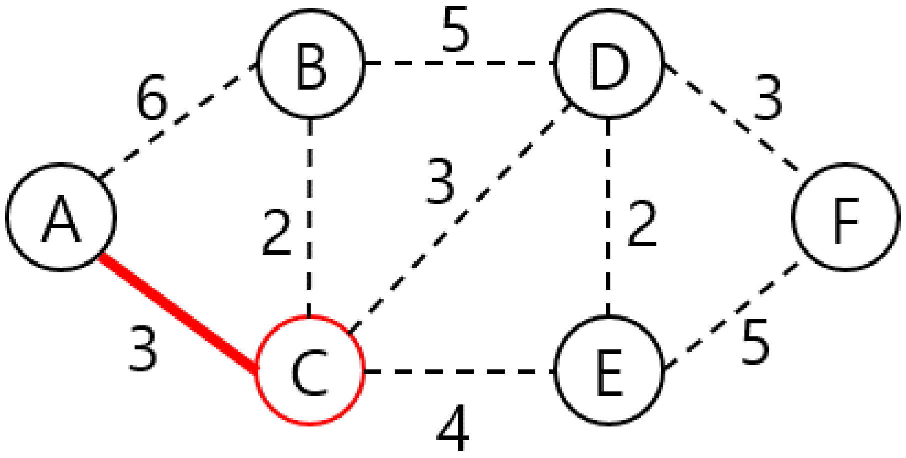


https://8iggy.tistory.com/159 참조

최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- 프림 (Prim) MST 알고리즘의 다른 예

  - A부터 시작
  - B, C 중에 C가 낮은 가중치

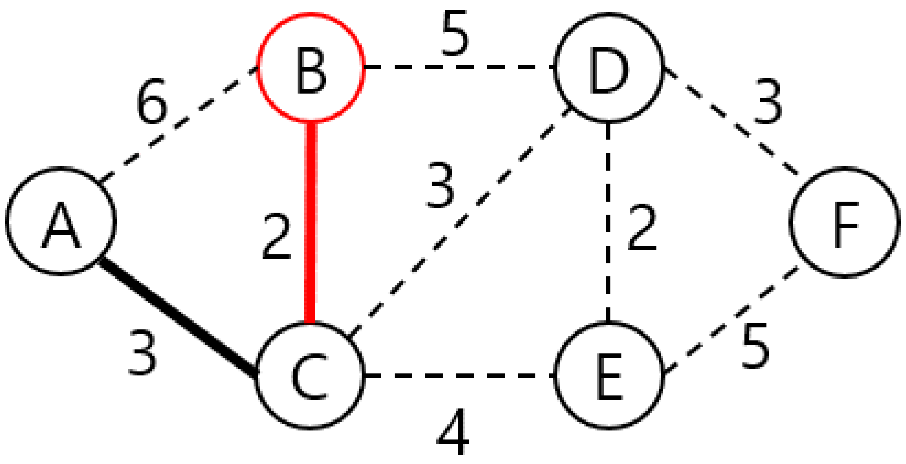


Nodes	[A, C]
Edges	[AC]

최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- 프림 (Prim) MST 알고리즘의 다른 예

A, C와 인접한 노드 중에 가장 낮은 가중치 : CB

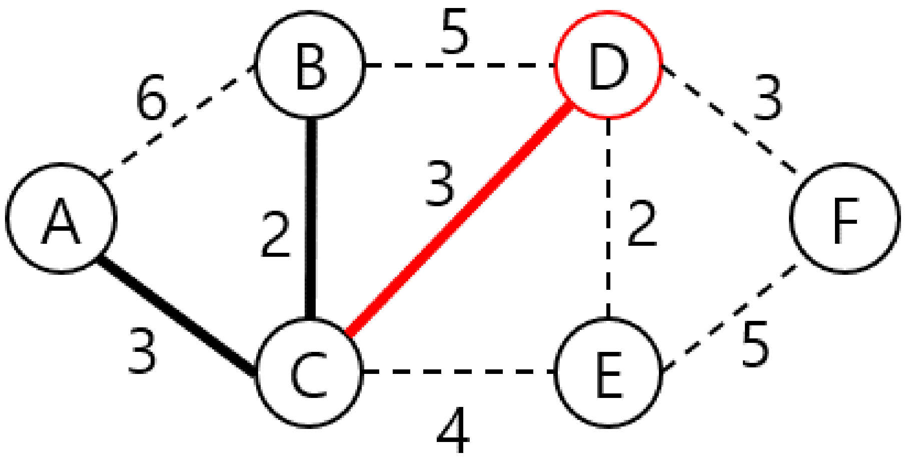


Nodes	[A, C, B]
Edges	[AC, CB]

최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- 프림 (Prim) MST 알고리즘의 다른 예

A, C, B와 인접한 노드 중에 가장 낮은 가중치 : CD

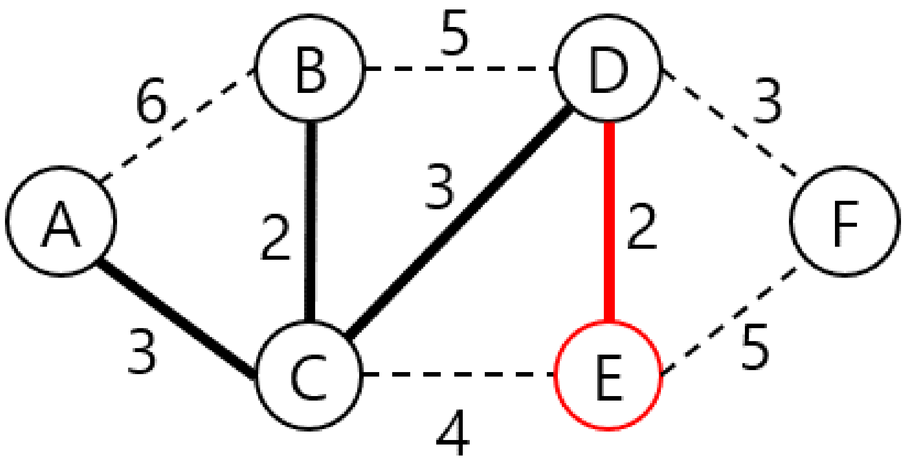


Nodes	[A, C, B, D]
Edges	[AC, CB, CD]

최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- 프림 (Prim) MST 알고리즘의 다른 예

A, C, B, D와 인접한 노드 중에 가장 낮은 가중치 : DE

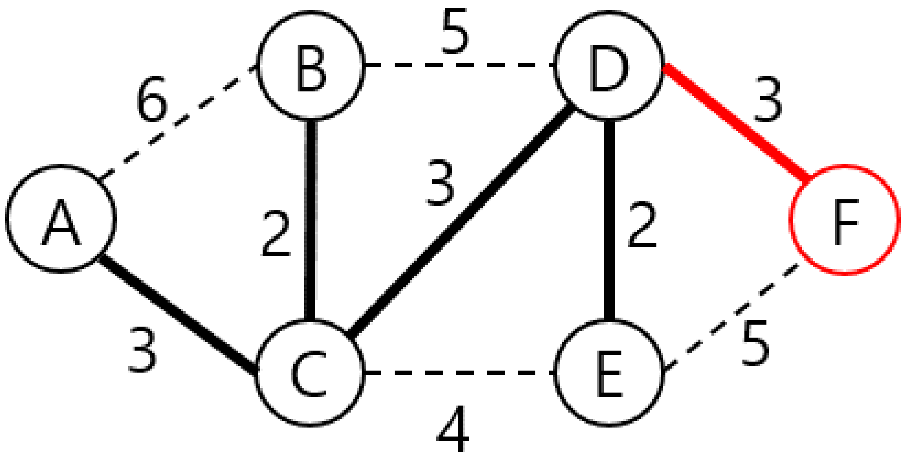


Nodes	[A, C, B, D, E]
Edges	[AC, CB, CD, DE]

최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- 프림 (Prim) MST 알고리즘의 다른 예

A, C, B, D, E와 인접한 노드 중에 가장 낮은 가중치 : DF



Nodes	[A, C, B, D, E, F]
Edges	[AC, CB, CD, DE, DF]
Cost	13

최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- 프림 (Prim)의 MST 알고리즘 시간 복잡도
  - while-loop가 (n-1)회 반복되고,
    - 1회 반복될 때, line 9에서 T에 속하지 않은 각 점 v에 대하여, D[v]가 최소인 점 v_{min} 을 찾는데 O(n) 시간 소요
    - 배열 D에서 (현재 T에 속하지 않은 점들에 대해서) 최솟값을 찾는 것이고, 배열의 크기는 n이기 때문
  - 프림 알고리즘의 시간 복잡도
    - (n-1) X O(n) = $O(n^2)$
    - 최소 힙(Binary Heap)을 사용하여 v_{min} 을 찾으면 O(mlogn), m은 간선의 수
      - 따라서 간선의 수가 O(n)이면 O(nlogn)

최소 신장 트리 (Minimum Spanning Tree) 알고리즘

- Kruskal과 Prim 알고리즘의 수행 과정 비교
  - Kruskal 알고리즘
    - 간선이 1개씩 T에 추가되는데, 이는 마치 n개의 점들이 각각의 트리인 상태에서 간선이 추가되면 2개의 트리가 1개의 트리로 합쳐지는 것과 같음
    - Kruskal 알고리즘은 이를 반복하여 1개의 트리인 T를 생성
    - n개의 트리들이 점차 합쳐져서 1개의 신장 트리가 만들어짐
  - 프림 알고리즘
    - T가 점 1개인 트리에서 시작되어 간선을 1개씩 추가
    - 1개의 트리가 자라나서 신장 트리가 됨

최단 경로 (Shortest Path) 찾기

- 최단 경로 (Shortest Path) 문제
  - 주어진 가중치 그래프에서 어느 한 출발점에서 또 다른 도착점까지의 최단 경로를 찾는 문제
- 최단 경로를 찾는 가장 대표적인 알고리즘
  - 다익스트라 (Dijkstra) 최단 경로 알고리즘
- 다익스트라 알고리즘
  - 주어진 출발점에서 시작
  - 출발점으로부터 최단 거리가 확정되지 않은 점들 중에서 출발점으로부터 가장 가까운 점을 추가하고, 그 점의 최단 거리를 확정

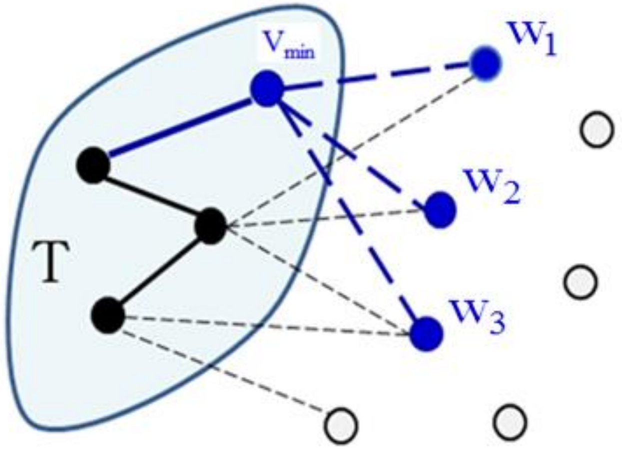
최단 경로 (Shortest Path) 찾기

- 최단 경로 찾기 알고리즘
  - ShortestPath(G,s)
    - 입력: 가중치 그래프 G=(V,E), |V|=n, |E|=m
    - 출력: 출발점 s로부터 (n-1)개의 점까지 각각 최단 거리를 저장한 배열 D

1. 배열 D를 ∞로 초기화. 단, D[s]=0으로 초기화 // 배열 D[v]에는 출발점 s로부터 점 v까지의 거리를 저장
2. while (s로부터의 최단 거리가 확정되지 않은 점이 있으면)
3. 현재까지 최단 거리가 확정되지 않은 각 점 v에 대해서 최소의 D[v]의 값을 가진 점 v_{min} 을 선택하고, s로부터 점 v_{min} 까지의 최단거리 D[v_{min}]을 확정
4. S로부터 현재보다 짧은 거리로 점 v_{min} 을 통해 우회 가능한 각 점 w에 대해서 D[w]를 갱신 // 간선 완화
5. return D

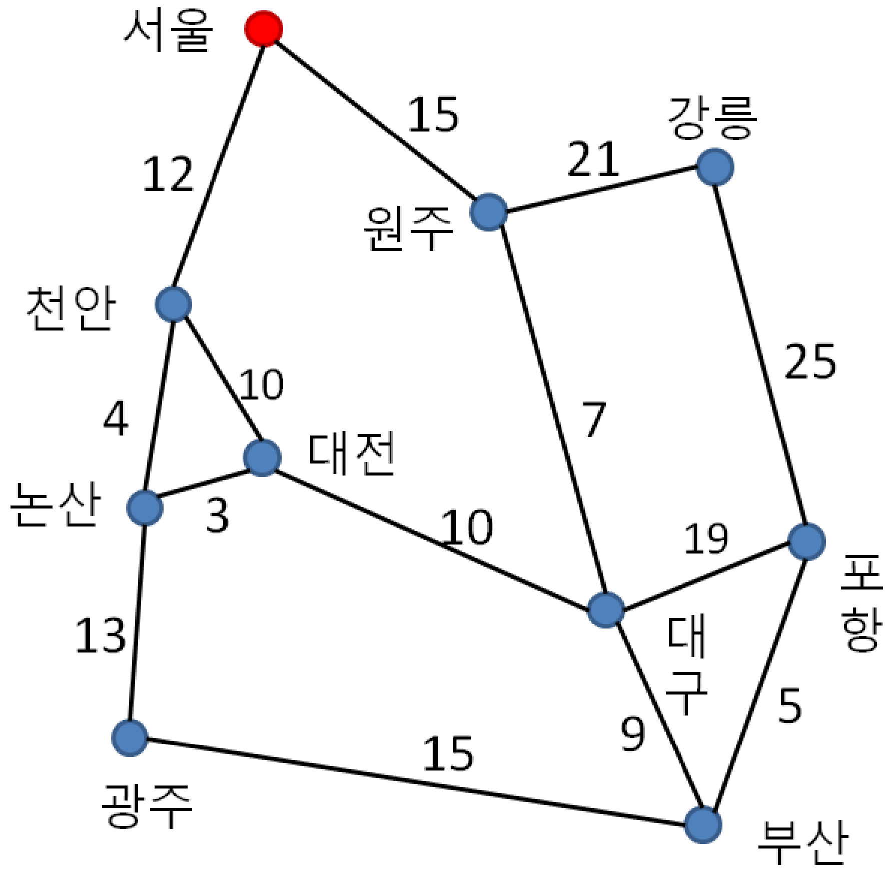
최단 경로 (Shortest Path) 찾기

- 간선 완화 (Edge Relaxation)
  - Line 4
    - V-T에 속한 점들 중 v_{min} 을 거쳐 감 (경유함)으로써 s로부터의 거리가 현재보다 더 짧아지는 점 w가 있으면, 그 점의 D[w]를 갱신
    - v_{min} 이 T에 포함된 상태에서 v_{min} 에 인접한 점 w_1, w_2, w_3 각각에 대해서 만일 (D[v_{min}]+간선 (v,w_i)의 가중치) < D[w_i]이면, D[w_i] = (D[v_{min}]+간선 (v, w_i)의 가중치)로 갱신



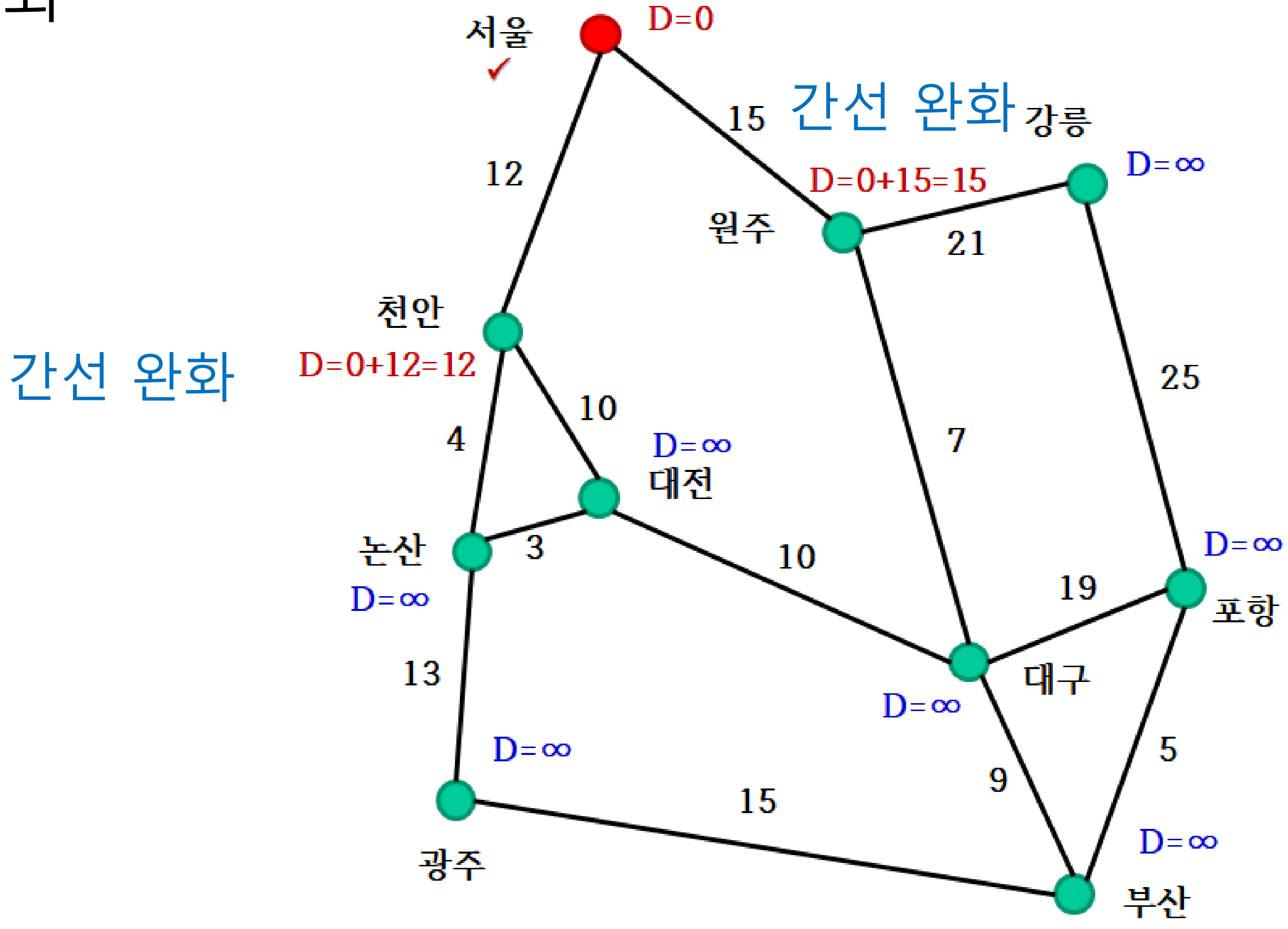
최단 경로 (Shortest Path) 찾기

- 최단 경로 찾기 알고리즘 수행 과정
  - 서울 확정



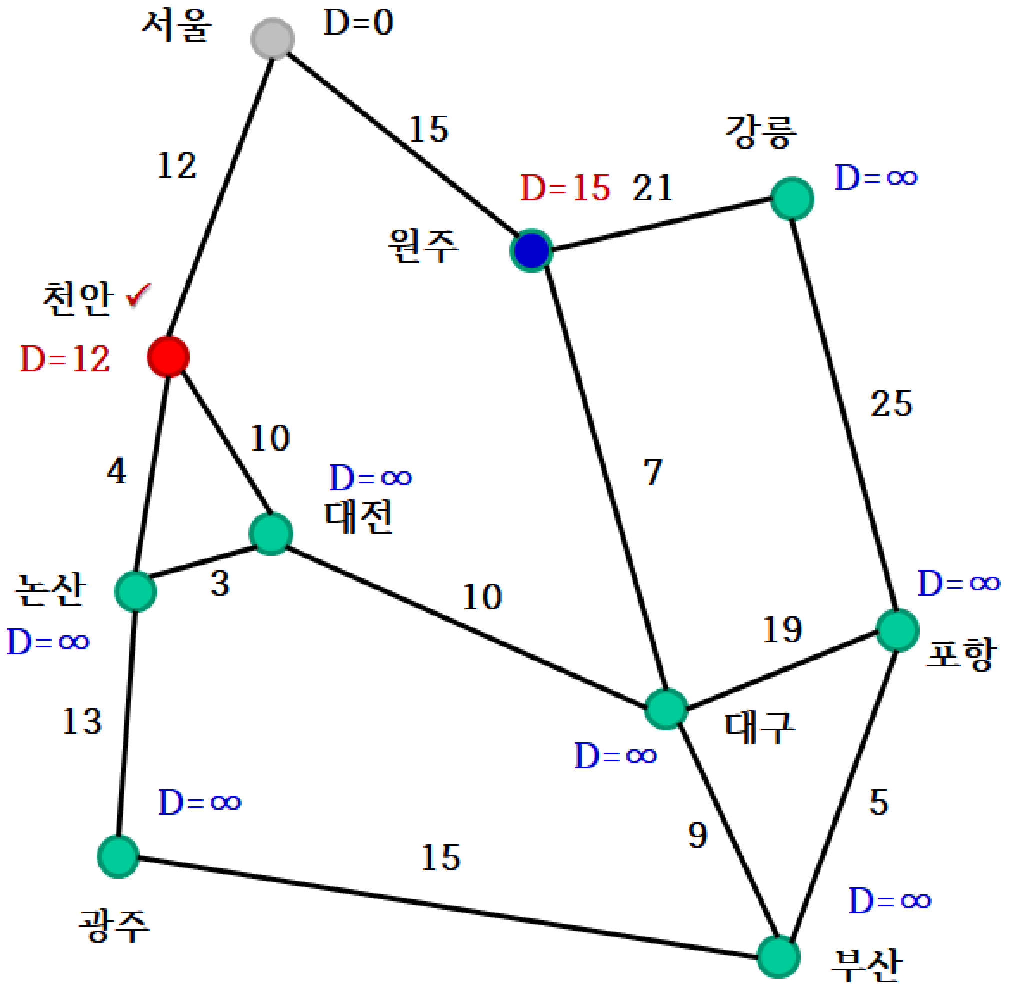
최단 경로 (Shortest Path) 찾기

- 최단 경로 찾기 알고리즘 수행 과정
  - 간선 완화



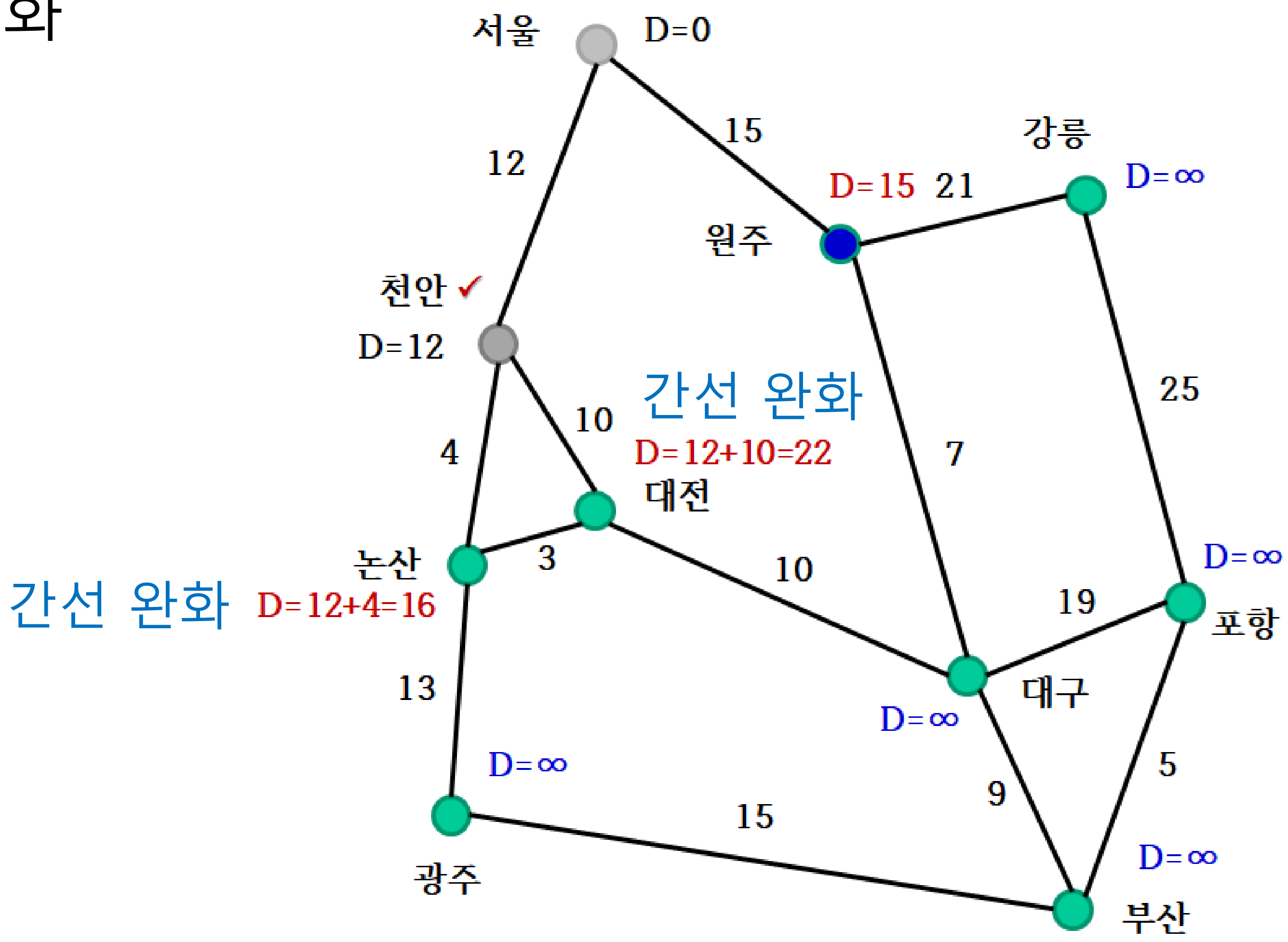
최단 경로 (Shortest Path) 찾기

- 최단 경로 찾기 알고리즘 수행 과정
  - 천안 확정



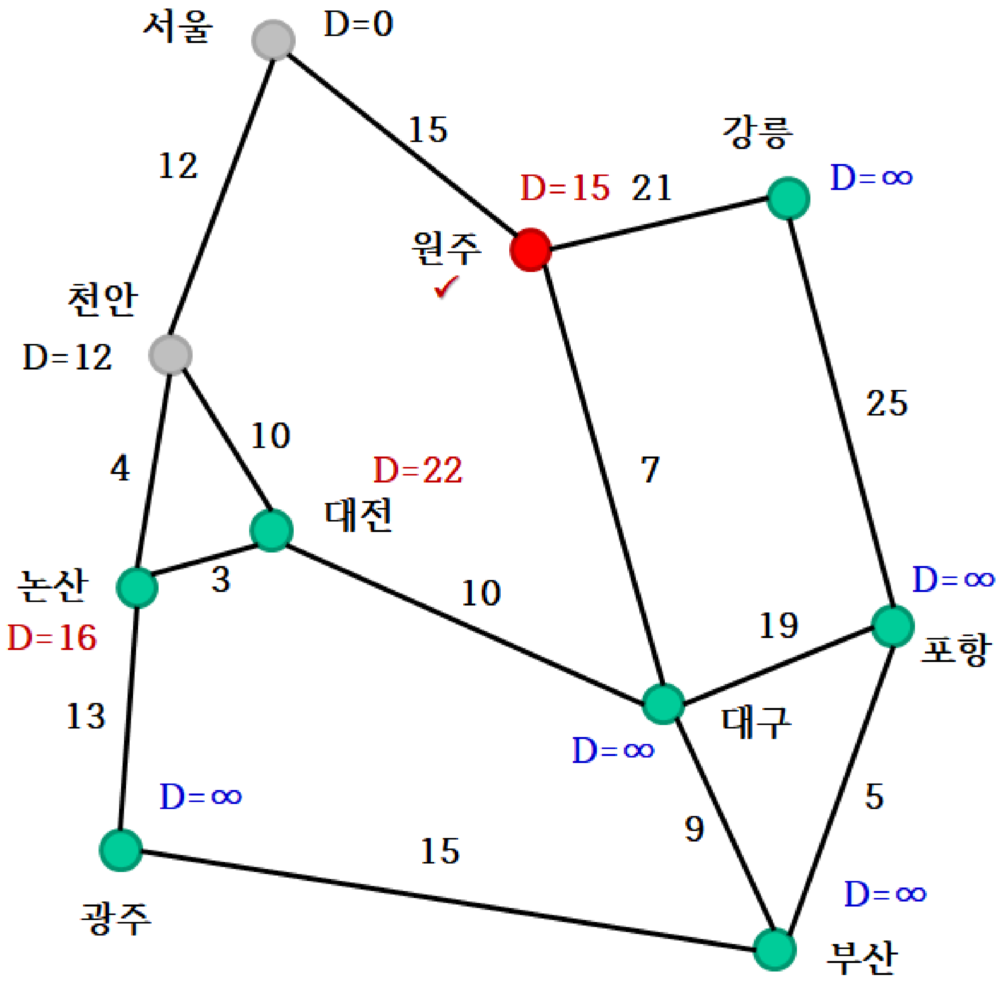
최단 경로 (Shortest Path) 찾기

- 최단 경로 찾기 알고리즘 수행 과정
  - 간선 완화



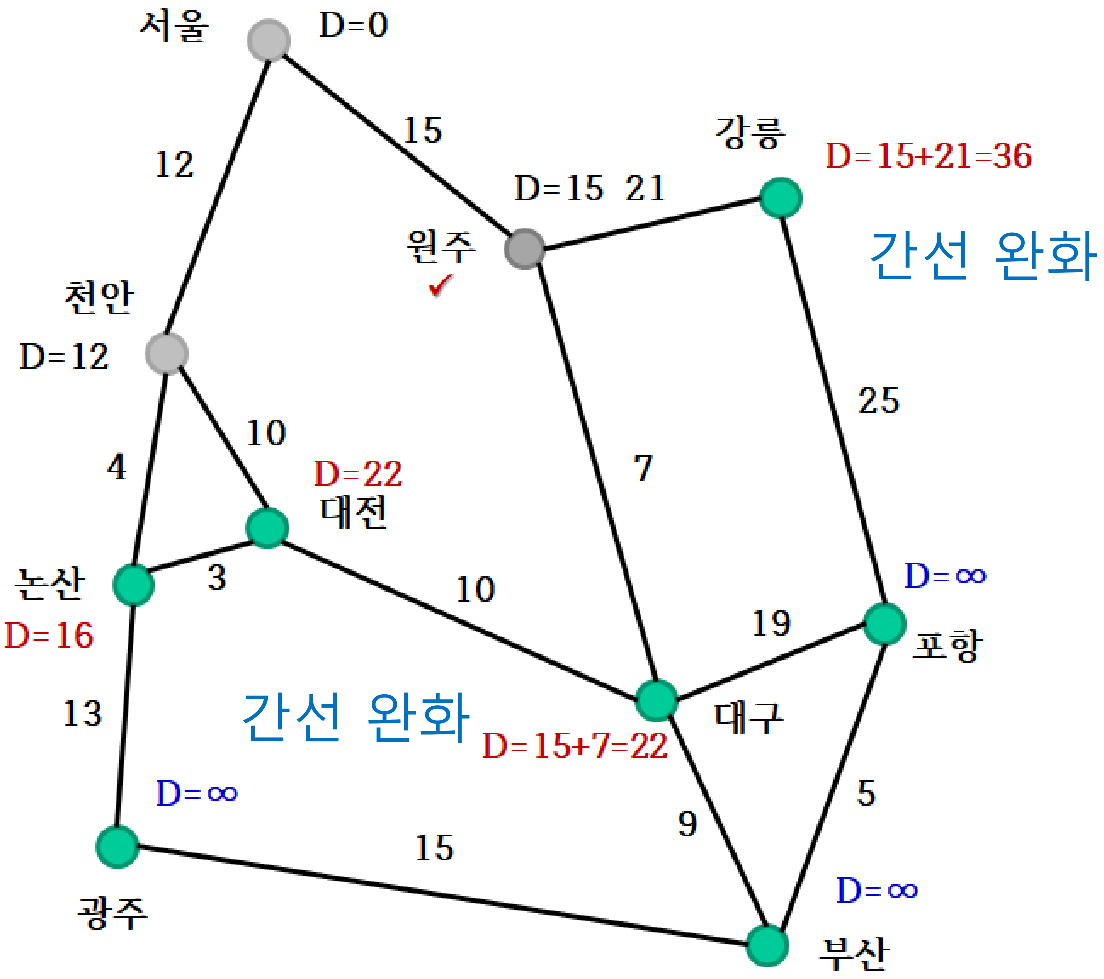
최단 경로 (Shortest Path) 찾기

- 최단 경로 찾기 알고리즘 수행 과정
  - 원주 확정



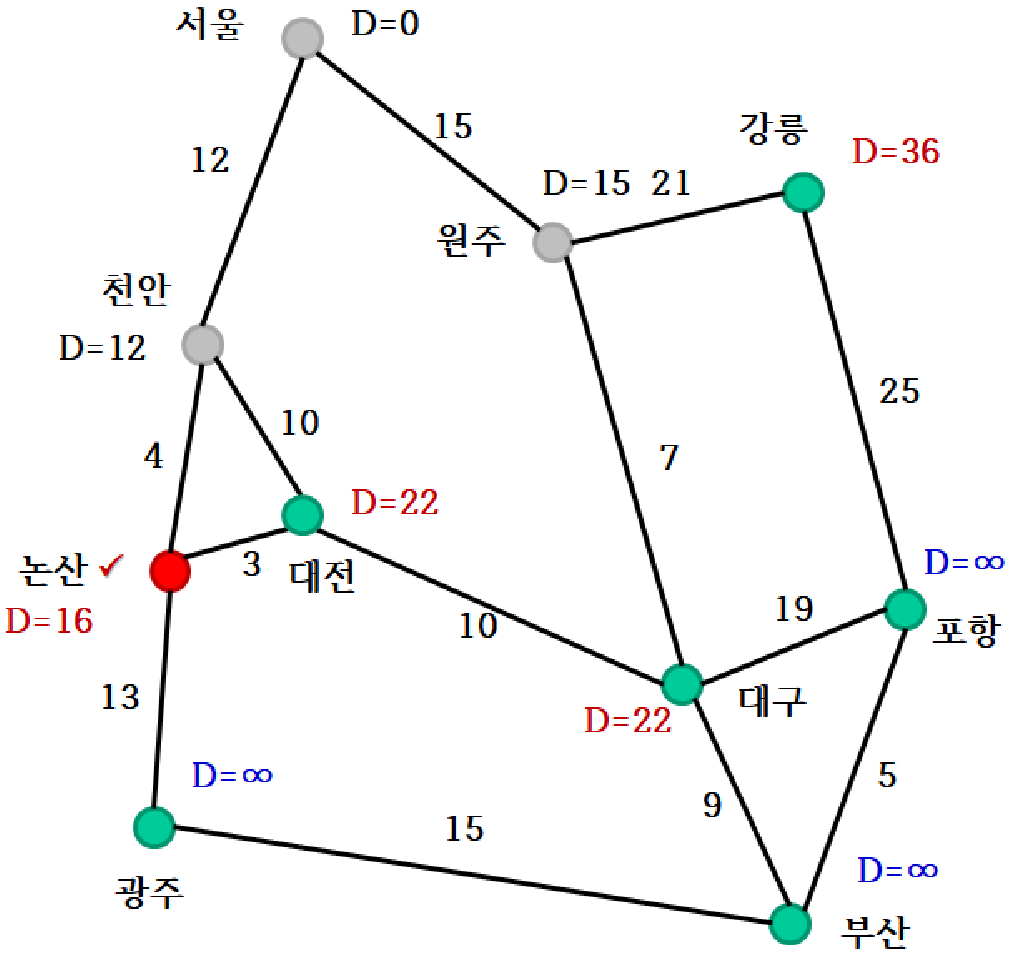
최단 경로 (Shortest Path) 찾기

- 최단 경로 찾기 알고리즘 수행 과정
  - 간선 완화



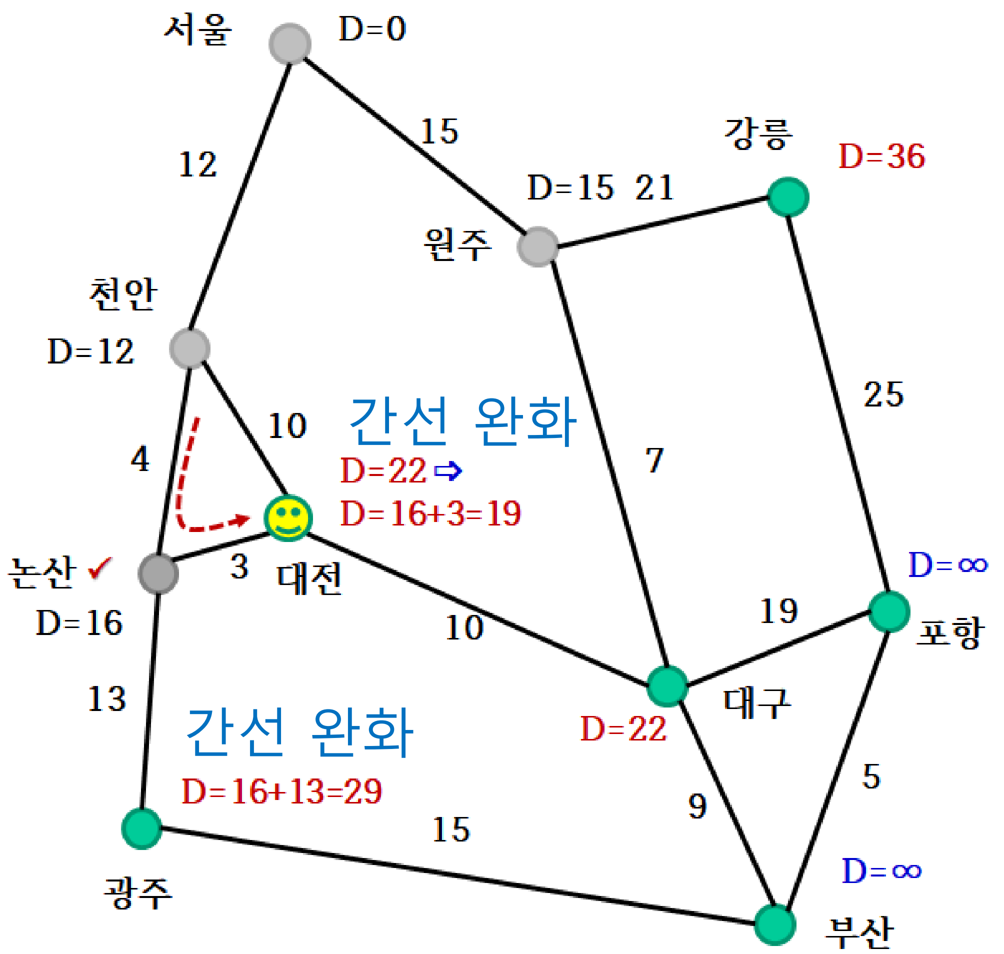
최단 경로 (Shortest Path) 찾기

- 최단 경로 찾기 알고리즘 수행 과정
  - 논산 확정



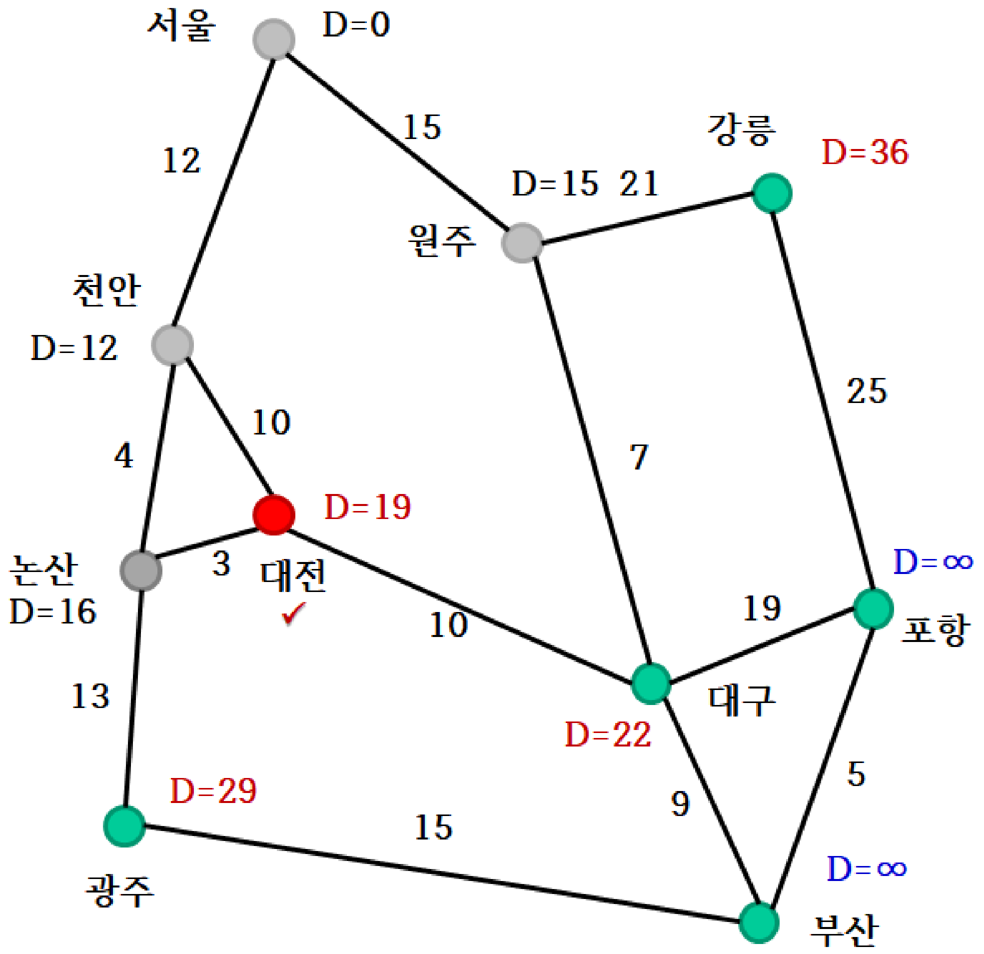
최단 경로 (Shortest Path) 찾기

- 최단 경로 찾기 알고리즘 수행 과정
  - 간선 완화



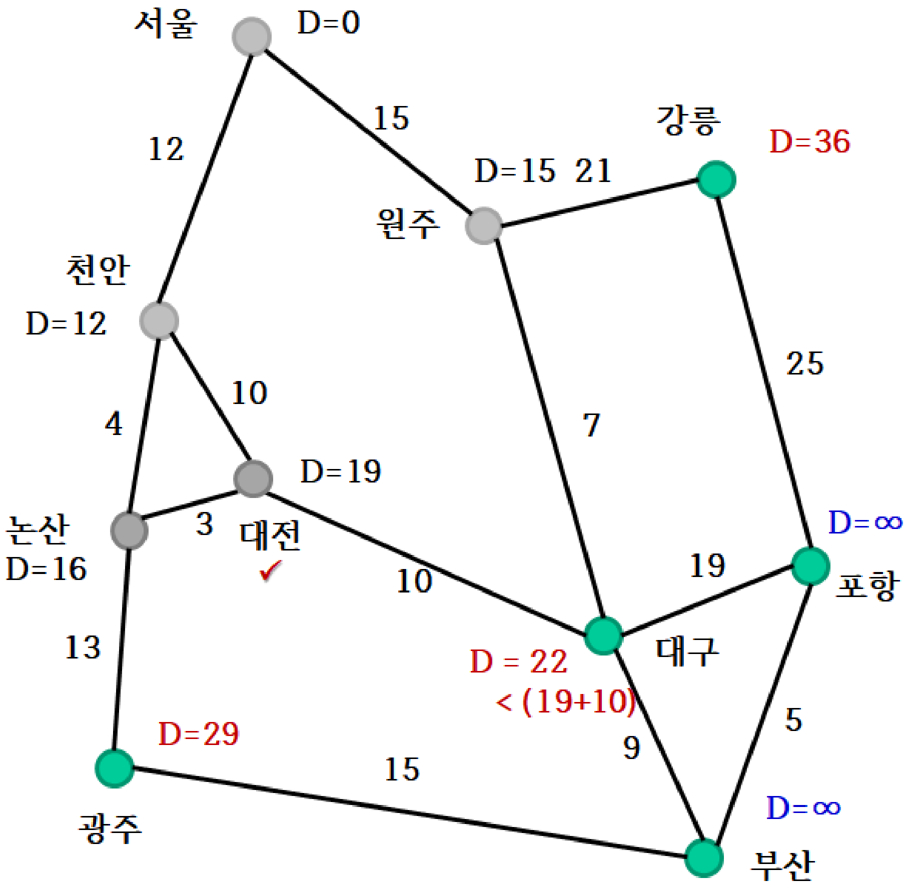
최단 경로 (Shortest Path) 찾기

- 최단 경로 찾기 알고리즘 수행 과정
  - 대전 확정



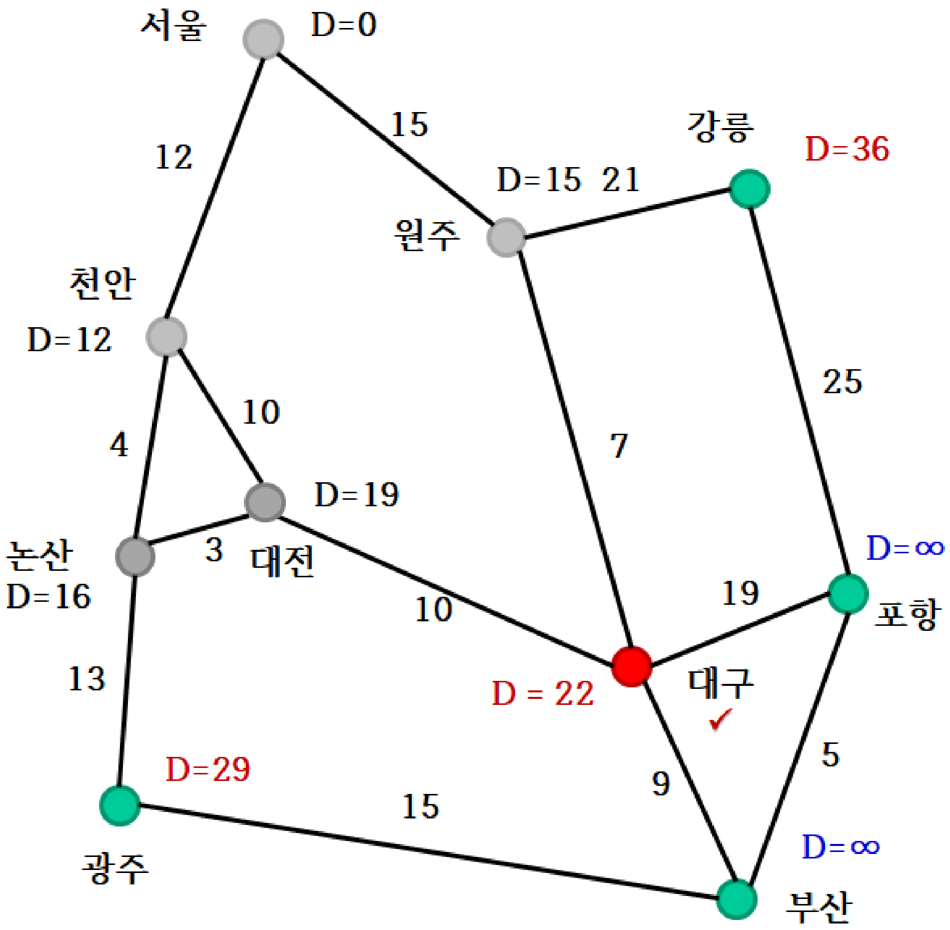
최단 경로 (Shortest Path) 찾기

- 최단 경로 찾기 알고리즘 수행 과정
  - 간선 완화 없음



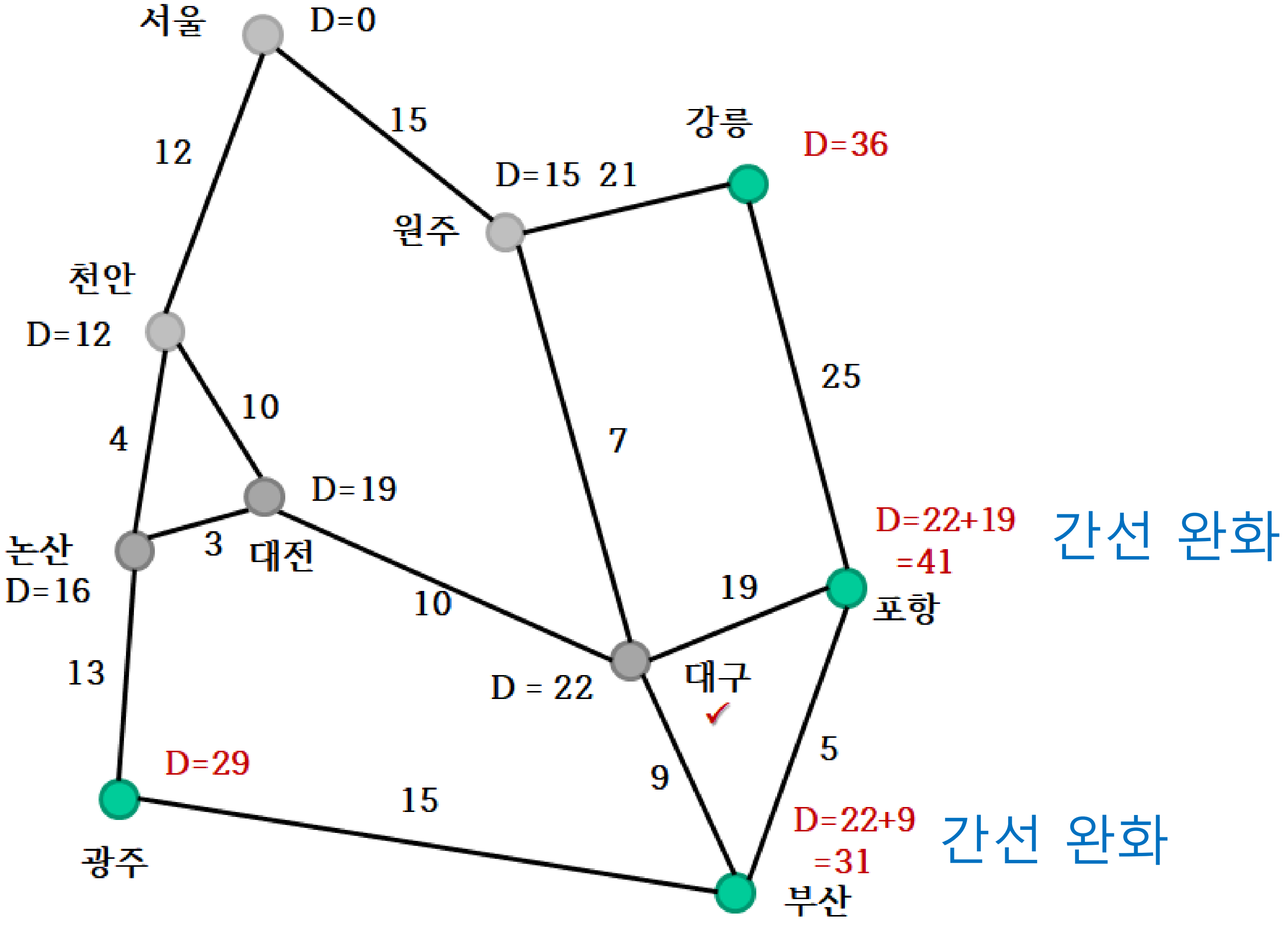
최단 경로 (Shortest Path) 찾기

- 최단 경로 찾기 알고리즘 수행 과정
  - 대구 확정



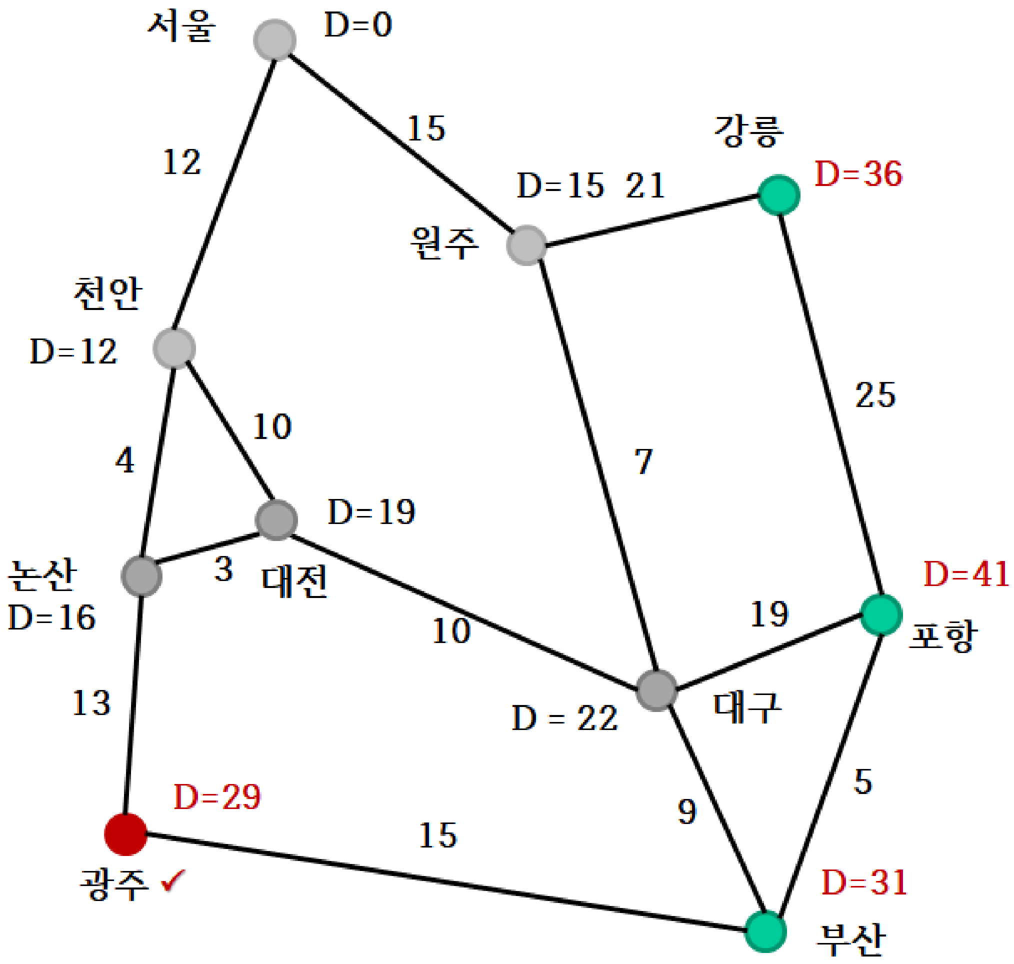
최단 경로 (Shortest Path) 찾기

- 최단 경로 찾기 알고리즘 수행 과정
  - 간선 완화



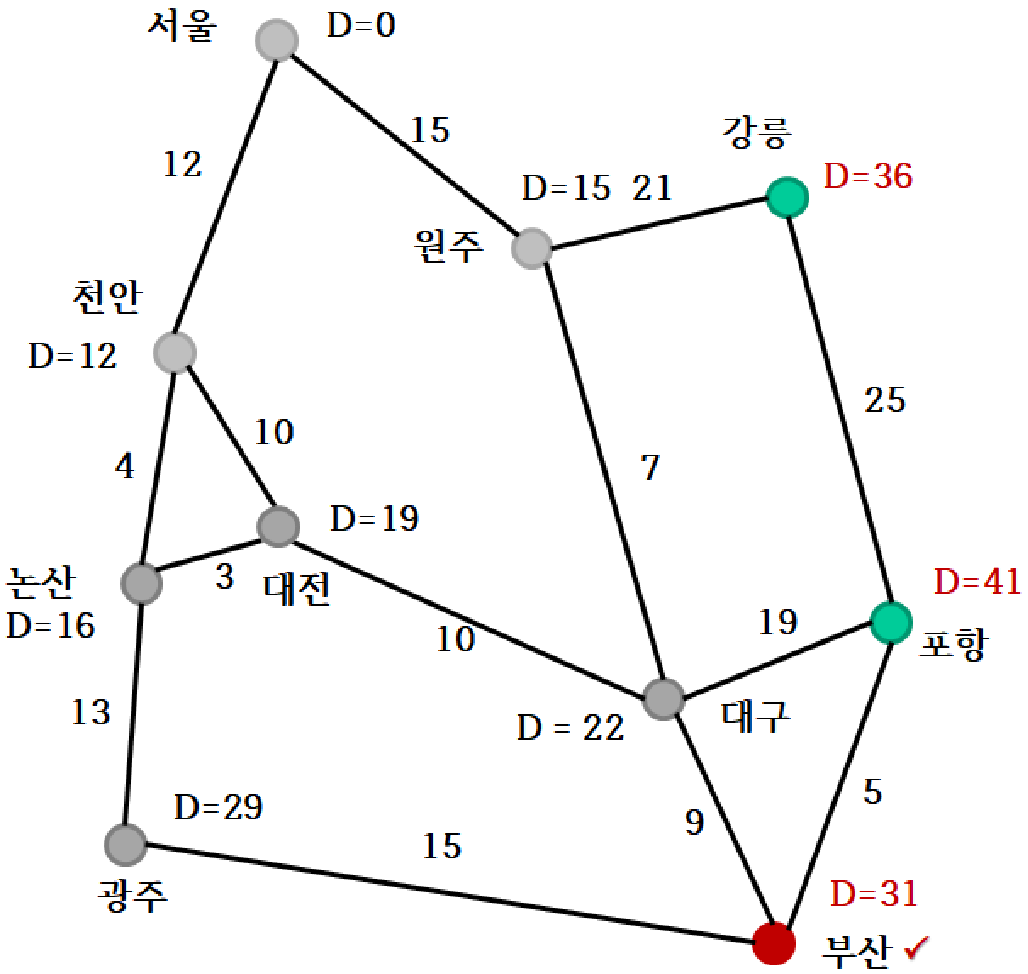
최단 경로 (Shortest Path) 찾기

- 최단 경로 찾기 알고리즘 수행 과정
  - 광주 확정



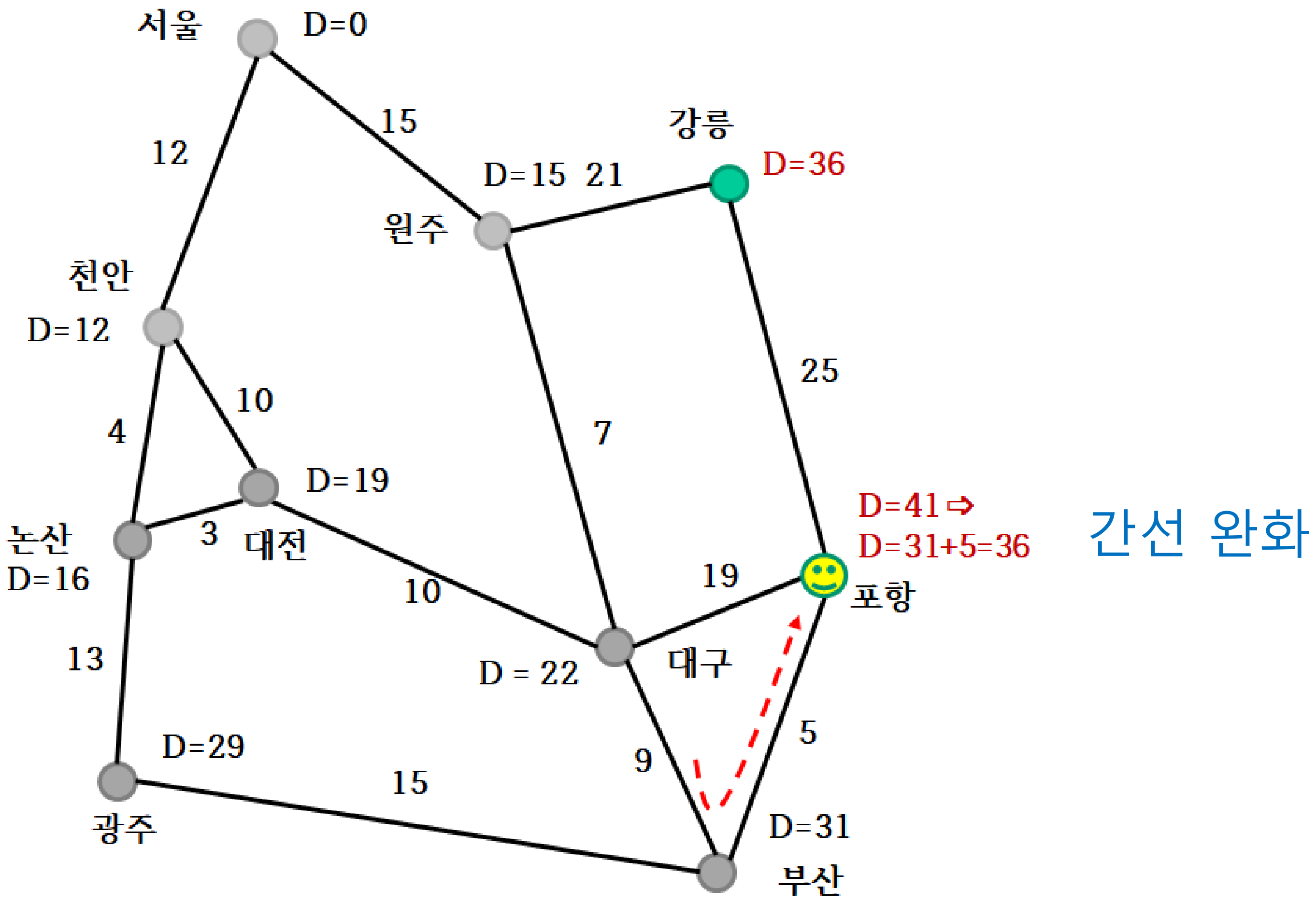
최단 경로 (Shortest Path) 찾기

- 최단 경로 찾기 알고리즘 수행 과정
  - 부산 확정



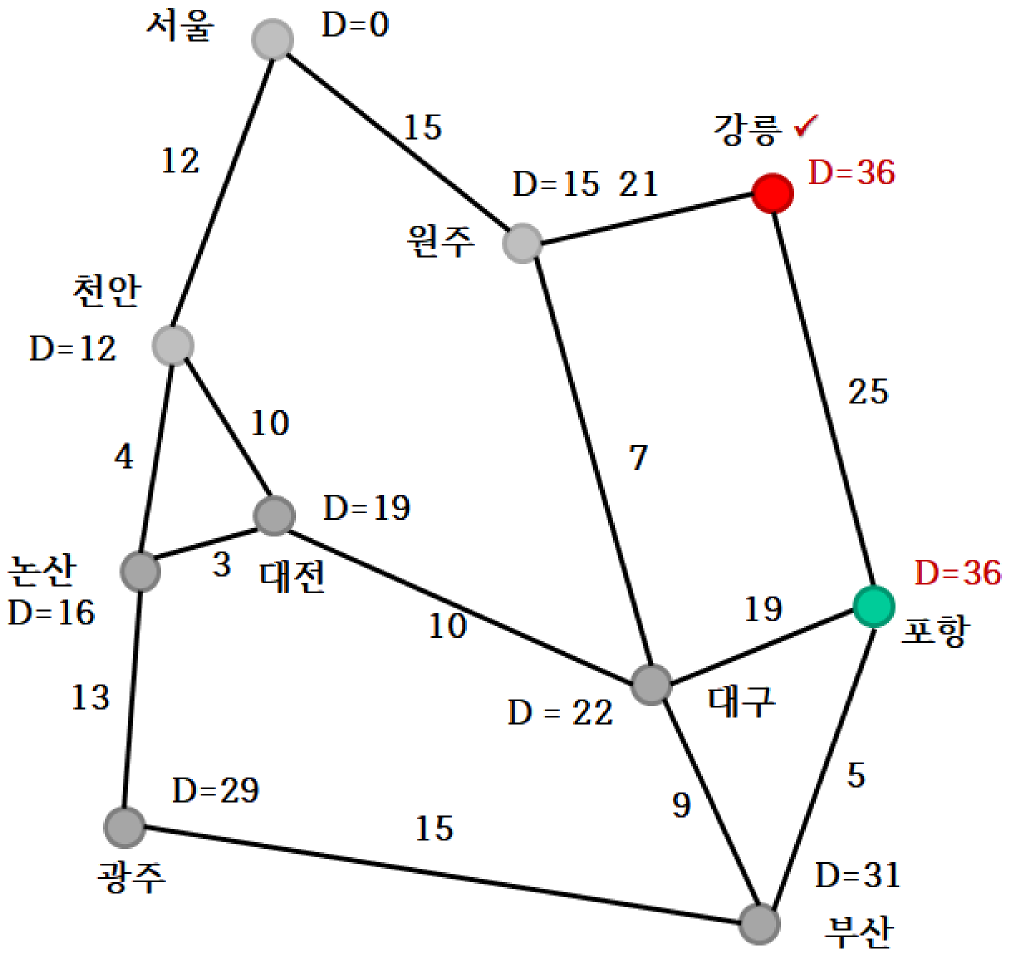
최단 경로 (Shortest Path) 찾기

- 최단 경로 찾기 알고리즘 수행 과정
  - 간선 완화



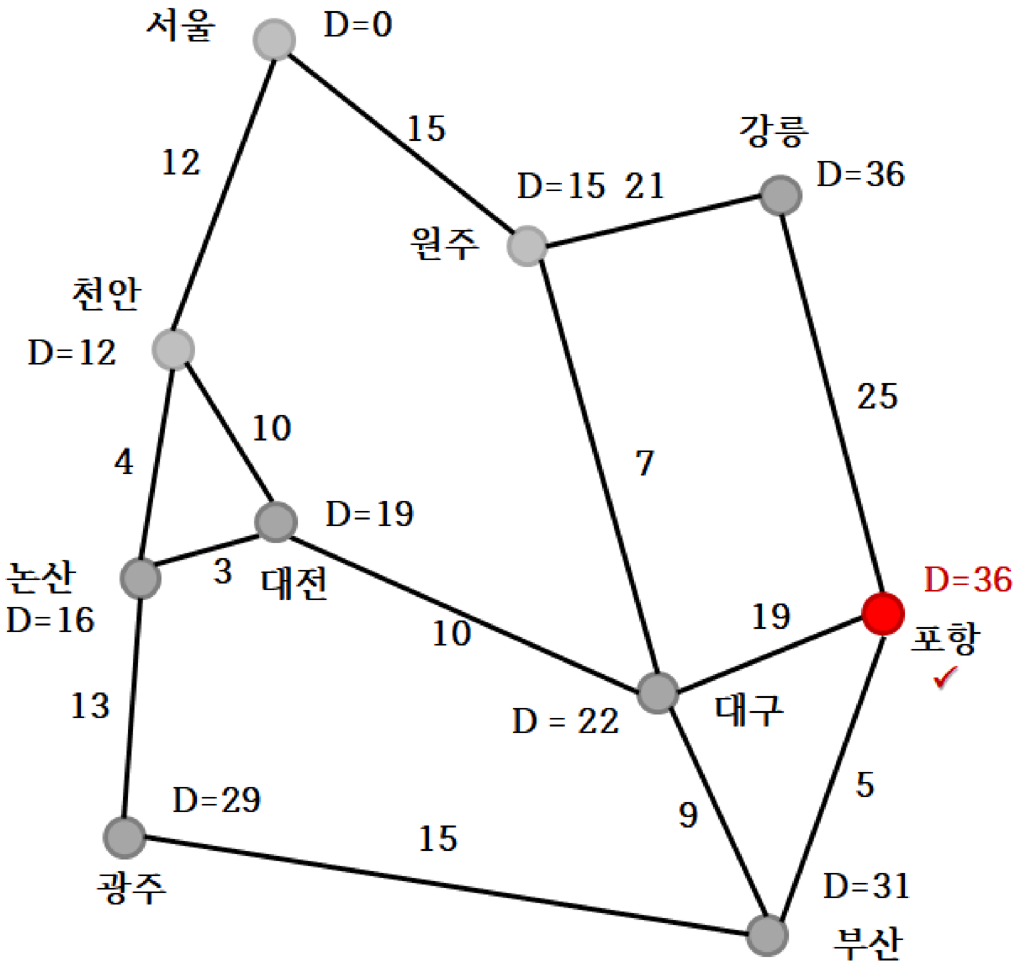
최단 경로 (Shortest Path) 찾기

- 최단 경로 찾기 알고리즘 수행 과정
  - 강릉 확정



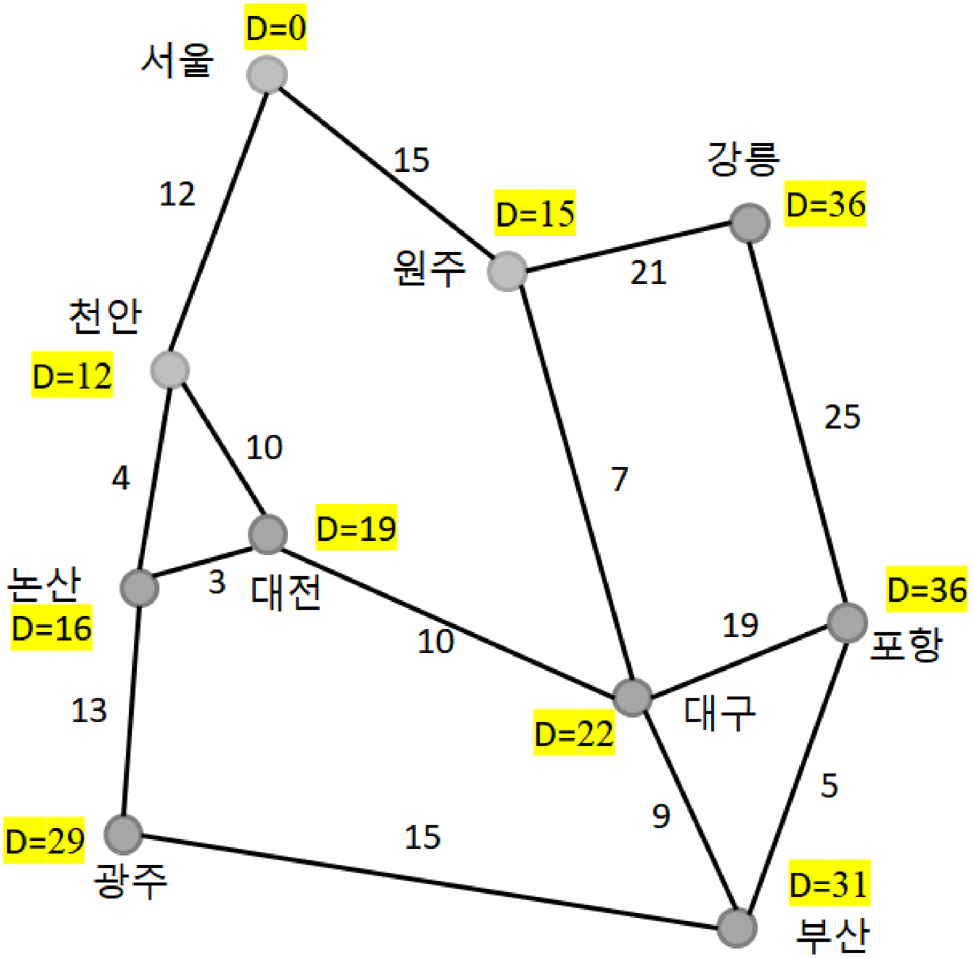
최단 경로 (Shortest Path) 찾기

- 최단 경로 찾기 알고리즘 수행 과정
  - 포항 확정



최단 경로 (Shortest Path) 찾기

- 최단 경로 찾기 알고리즘 수행 과정
  - 알고리즘 수행 결과



최단 경로 (Shortest Path) 찾기

- 시간 복잡도
  - while-loop가 (n-1)회 반복되고, 1회 반복될 때
    - Line 3에서 최소의 D[v]를 가진 점 v_{min}을 찾는데 O(n) 시간 소요
      - 배열 D에서 최솟값을 찾기 때문
    - Line 4에서도 v_{min}에 연결된 점의 수가 최대 (n-1)개이므로, 각 D[w]를 갱신하는데 걸리는 시간은 O(n)
  - ShortestPath의 시간 복잡도
    - (n-1) X {O(n)+O(n)} = $O(n^2)$
    - 프림 알고리즘과 같이 최소 힙 (Binary Heap)을 사용하면 O(mlogn) (m: 간선의 수)
      - 따라서 간선의 수가 O(n)이면 시간 복잡도는 O(nlogn)