제4장 그리디 알고리즘 (1)

과 목 명 정 보 처 리 알 고 리 즘

담당교수 김 성 훈

경북대학교 과학기술대학 소프트웨어학과

이 장에서 배울 내용

1. 그리디 알고리즘 기본개념
2. 동전 거스름돈
3. 최소 신장 트리(Minimum Spanning Tree)
4. 최단 경로 찾기(Shortest Path Problem)
5. 부분배낭문제(Fractional Knapsack Problem)
6. 집합 커버 문제(Set Cover Problem)
7. 작업 스케쥴링 문제(Job Scheduling)
8. 허프만 압축(Huffman Coding)

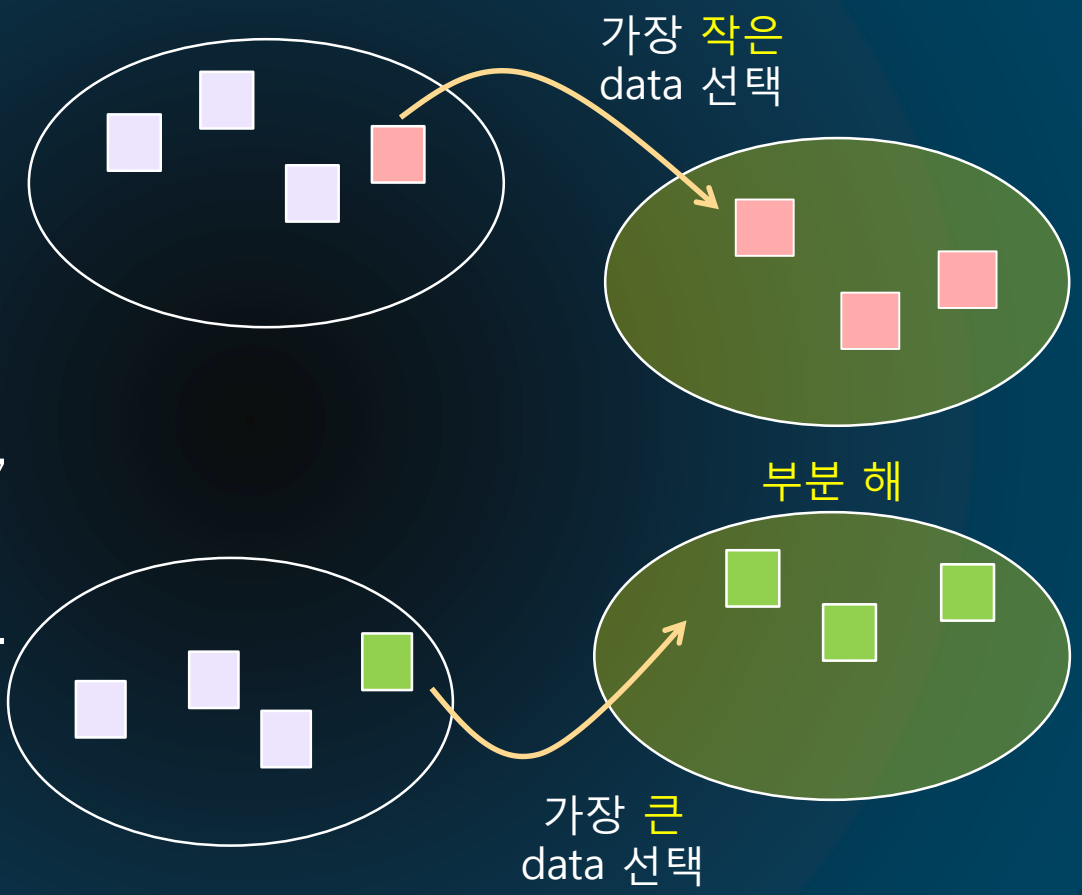
그리디(Greedy) 알고리즘의 기본 개념

- 그리디 알고리즘은 최적화 문제를 해결한다.
  - 최적화 (optimization) 문제: 가능한 해들 중에서 가장 좋은 (최대 또는 최소) 해를 찾는 문제
  - 욕심쟁이 방법, 탐욕적 방법, 탐욕 알고리즘, 등으로 불리기도 함.

- 그리디 알고리즘은 (입력)데이터 간의 관계를 고려하지 않고 수행 과정에서 '욕심내어' 최소값 또는 최대값을 가진 데이터를 선택한다.
  - 이러한 선택을 '근시안적'인 선택이라고 말하기도 한다.

그리디(Greedy) 알고리즘의 기본 개념(2)

- 그리디 알고리즘은
  근시안적인 선택으로
  부분적인 최적해를 찾고,
  이들을 모아서
  문제의 최적해를 얻는다.



그리디 알고리즘 수행 과정

그리디(Greedy) 알고리즘의 기본 개념(3)

- 그리디 알고리즘은 일단 한번 선택하면, 이를 절대로 번복하지 않는다.
  - 즉, 선택한 데이터를 버리고 다른 것을 취하지 않는다.
  - 이러한 특성 때문에 대부분의 그리디 알고리즘들은 매우 단순하며, 또한 제한적인 문제들만이 그리디 알고리즘으로 해결된다.
- 8장에서 다루는 대부분의 근사 알고리즘들은 그리디 알고리즘들이고, 9장의 해를 탐색하는 기법들 중의 하나인 분기 한정 기법도 그리디 알고리즘의 일종이다.

4.1 동전 거스름돈

- 동전 거스름돈 (Coin Change) 문제를 해결하는 가장 간단하고 효율적인 방법은 남은 액수를 초과하지 않는 조건하에 '욕심내어' 가장 큰 액면의 동전을 취하는 것이다.

- 다음은 동전 거스름돈 문제의 최소 동전 수를 찾는 그리디 알고리즘이다. 단, 동전의 액면은 500원, 100원, 50원, 10원, 1원이다.

동전 거스름돈 알고리즘

CoinChange(W)

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 각각의 동전 카운트
2. while (change ≥ 500)
   change = change-500, n500++ // 500원짜리 동전 수를 1 증가
3. while (change ≥ 100)
   change = change-100, n100++ // 100원짜리 동전 수를 1 증가
4. while (change ≥ 50)
   change = change-50, n50++ // 50원짜리 동전 수를 1 증가
5. while (change ≥ 10)
   change = change-10, n10++ // 10원짜리 동전 수를 1 증가
6. while (change ≥ 1)
   change = change-1, n1++ // 1원짜리 동전 수를 1 증가
7. return (n500+n100+n50+n10+n1) // 총 동전 수를 리턴한다.

동전 거스름돈 알고리즘의 특성

- CoinChange 알고리즘은 남아있는 거스름돈인 change에 대해 가장 높은 액면의 동전을 거스르며, 500원짜리 동전을 처리하는 line 2에서는 100원짜리, 50원짜리, 10원짜리, 1원짜리 동전을 몇 개씩 거슬러 주어야 할 것인지에 대해서는 전혀 고려하지 않는다.

- 이것이 바로 그리디 알고리즘의 근시안적인 특성이다.

동전 거스름돈 알고리즘의 문제점

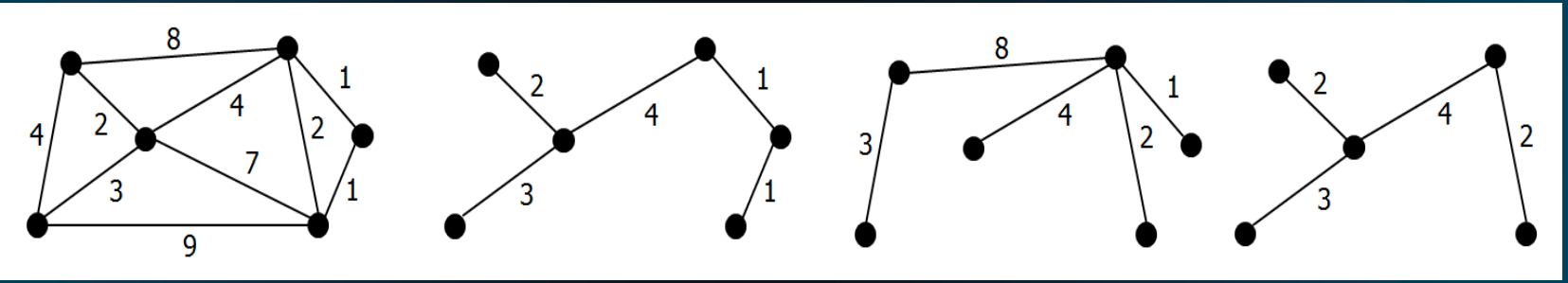
- 그런데 만일 한국은행에서 160원짜리 동전을 추가로 발행한다면, CoinChange 알고리즘이 항상 최소 동전 수를 계산할 수 있을까?
- 거스름돈이 200원이라면, CoinChange 알고리즘은 160원짜리 동전 1개와 10원짜리 동전 4개로서 총 5개를 리턴한다.

- CoinChange 알고리즘은 항상 최적의 답을 주지 못한다.
- 5장에서는 어떤 경우에도 최적해를 찾는 동전 거스름돈을 위한 동적 계획 알고리즘을 소개한다.

4.2 최소 신장 트리

- 최소 신장 트리 (Minimum Spanning Tree):
  - 주어진 가중치 그래프에서 사이클이 없이 모든 점들을 연결시킨 트리들 중 선분들의 가중치 합이 최소인 트리

(예시)

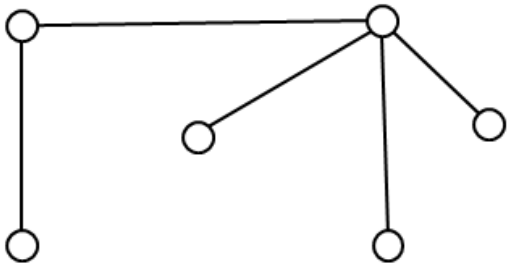


(a) (b) (c) (d)

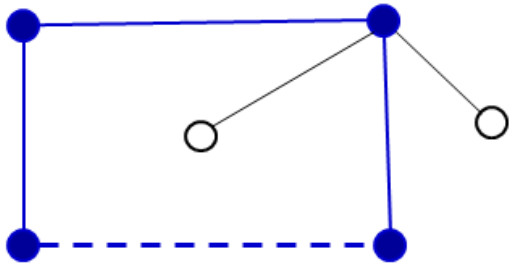
- (a)주어진 가중치 그래프, (b) 최소 신장 트리 (c),(d)는 최소 신장 트리 아님
- (c)는 가중치의 합이 (b)보다 크고,
  (d)는 트리가 주어진 그래프의 모든 노드를 포함하지 않고 있다.

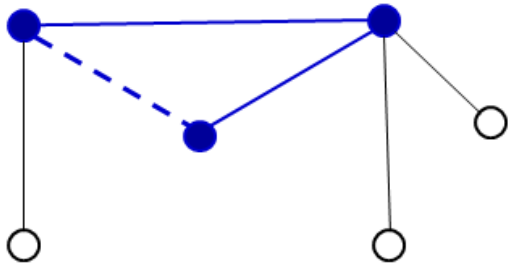
신장트리(Spanning Tree)의 특징

- 주어진 그래프의 신장 트리를 찾으려면 사이클이 없도록 모든 점을 연결시키면 된다.
  - 그래프의 점의 수가 n이면, 신장 트리에는 정확히 (n-1)개의 선분이 있다.
- 트리에 선분을 하나 추가시키면, 반드시 사이클이 만들어진다.



트리





점선으로 된 선분을 추가하여 만들어진 사이클

2가지 대표적인 알고리즘

- 최소 신장 트리를 찾는 대표적인 그리디 알고리즘으로는 크루스컬 (Kruskal)과 프림 (Prim) 알고리즘이 있다.
- 알고리즘의 입력은 1개의 연결된 구성요소(connected component)로 된 가중치 그래프이다.
- Kruskal 알고리즘은 가중치가 가장 작은 선분이 사이클을 만들지 않을 때에만 '욕심내어' 그 선분을 추가시킨다.

다음은 Kruskal의 최소 신장 트리 알고리즘이다.

Kruskal의 최소 신장 트리 알고리즘

KruskalMST(G)

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

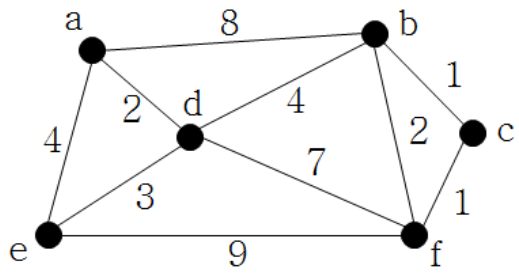
1. 가중치의 오름차순으로 선분들을 정렬한다. 정렬된 선분 리스트를 L이라고 하자.
2. T=Ø // 트리 T를 초기화시킨다.
3. while (T의 선분 수 < n-1) {
4. L에서 가장 작은 가중치를 가진 선분 e를 가져오고, e를 L에서 제거한다.
5. if (선분 e가 T에 추가되어 사이클을 만들지 않으면)
6. e를 T에 추가시킨다.
7. else // e가 T에 추가되어 사이클이 만들어지는 경우
8. e를 버린다.
   }
9. return 트리 T // T는 최소 신장 트리이다.

Kruskal 알고리즘의 설명(생략)

- Line 1: 모든 선분들을 가중치의 오름차순으로 정렬한다. 정렬된 선분들의 리스트를 L이라고 하자.
- Line 2: T를 초기화시킨다. 즉, T에는 아무 선분도 없는 상태에서 시작된다.
- Line 3~8의 while-루프는 T의 선분 수가 (n-1)이 될 때까지 수행되는데 1번 수행될 때마다 L에서 가중치가 가장 작은 선분 e를 가져온다. 단, 가져온 선분 e는 L에서 삭제되어 다시는 고려되지 않는다.
- Line 5~8: 가져온 선분 e를 T에 추가되어 사이클을 만들지 않으면 e를 T에 추가시키고, 사이클을 만들면 선분 e를 버린다. 왜냐하면 모든 노드들이 연결되어 있으면서 사이클이 없는 그래프가 신장 트리이기 때문이다.

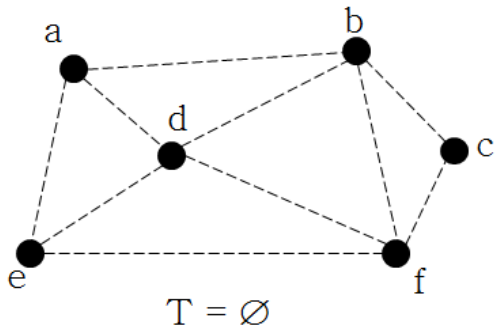
Kruskal 알고리즘의 실행과정

- 다음의 그래프에서 KruskalMST 알고리즘이 최소 신장 트리를 찾는 과정을 살펴보자.



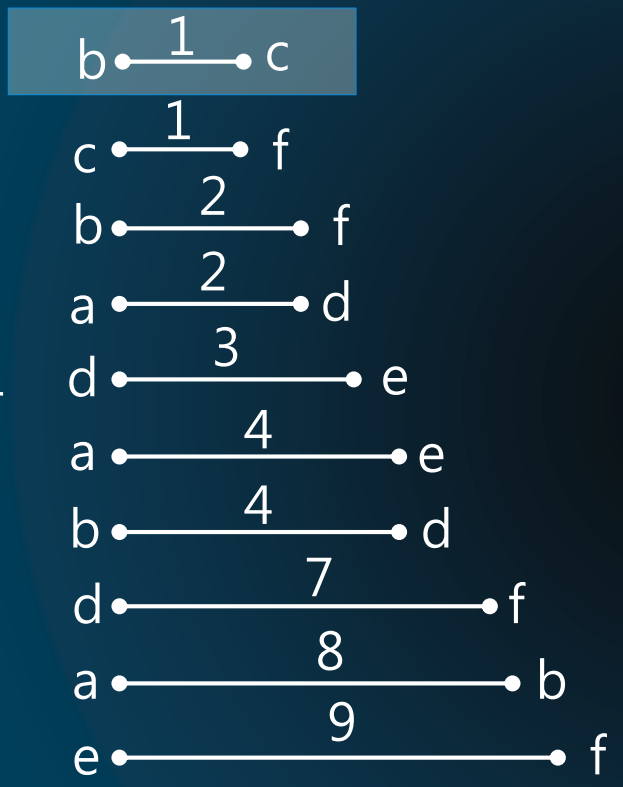
정렬된 리스트 L

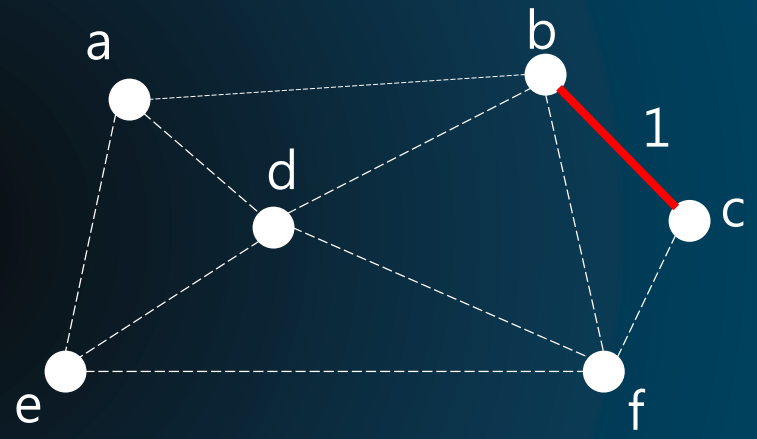
b 1 c
c 1 f
b 2 f
a 2 d
d 3 e
a 4 e
b 4 d
d 7 f
a 8 b
e 9 f



Kruskal 알고리즘의 실행과정(2)

리스트
L

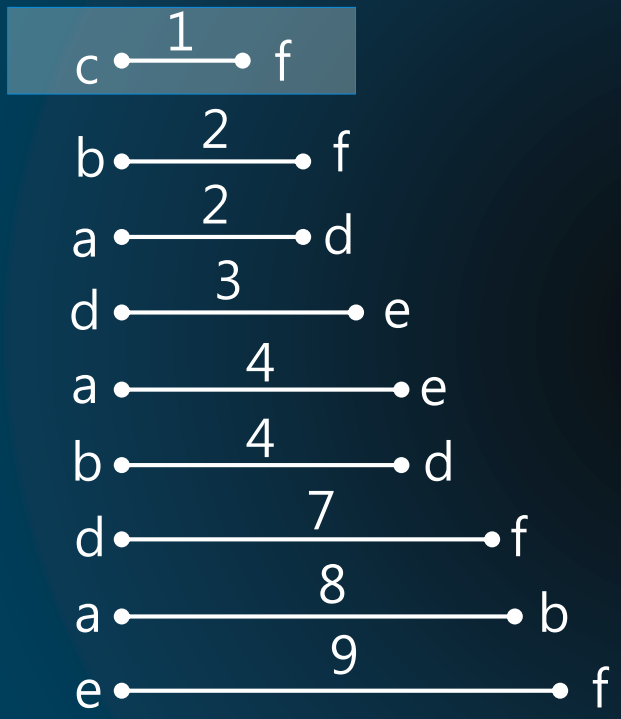


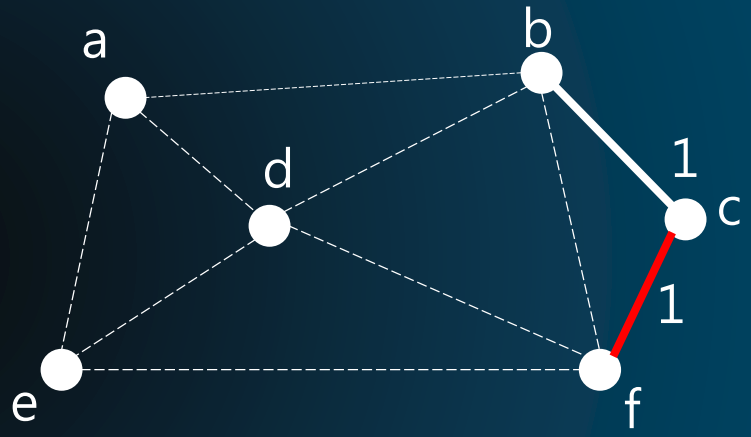


선분 (b,c) 추가

Kruskal 알고리즘의 실행과정(3)

리스트
L

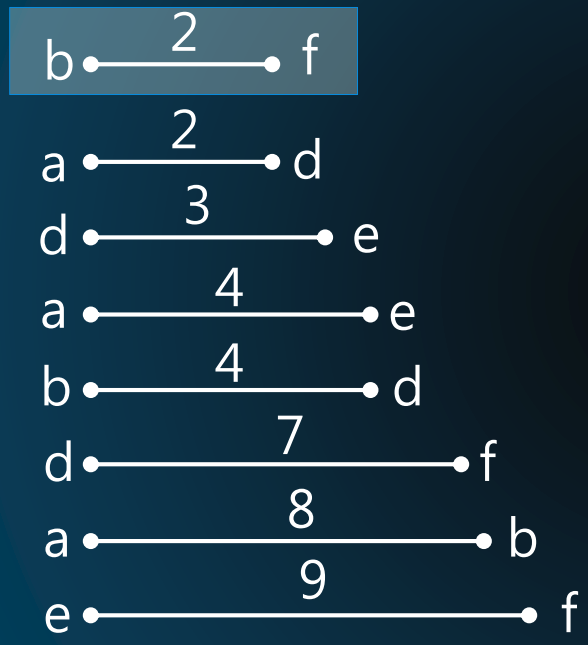


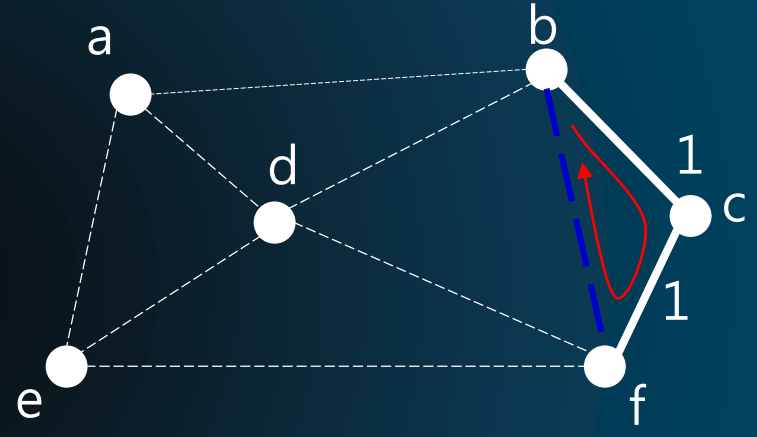


선분 (c,f) 추가

Kruskal 알고리즘의 실행과정(4)

리스트
L



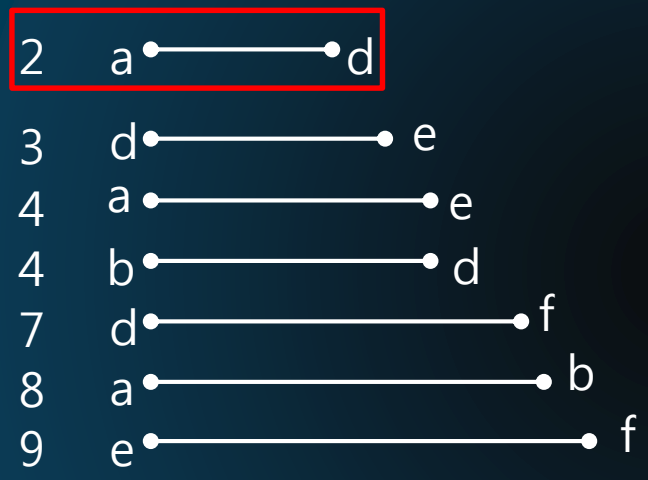


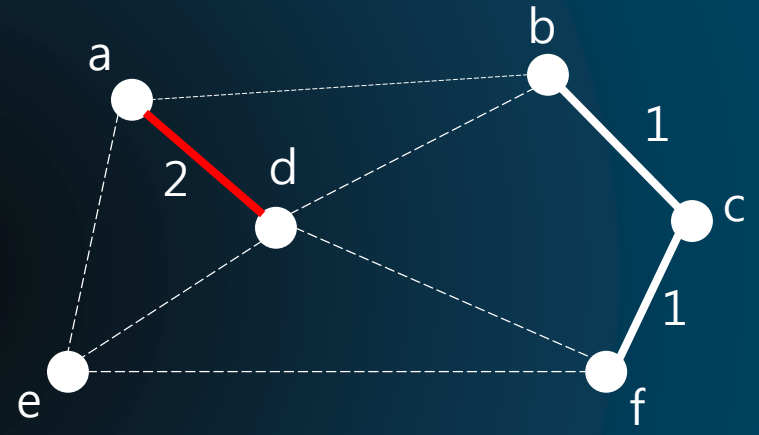
사이클 b-c-f-b

선분 (b,f) 버림

Kruskal 알고리즘의 실행과정(5)

정렬된 리스트 L

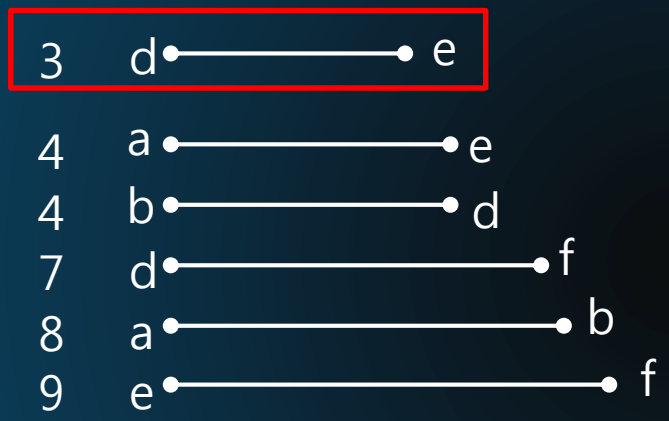


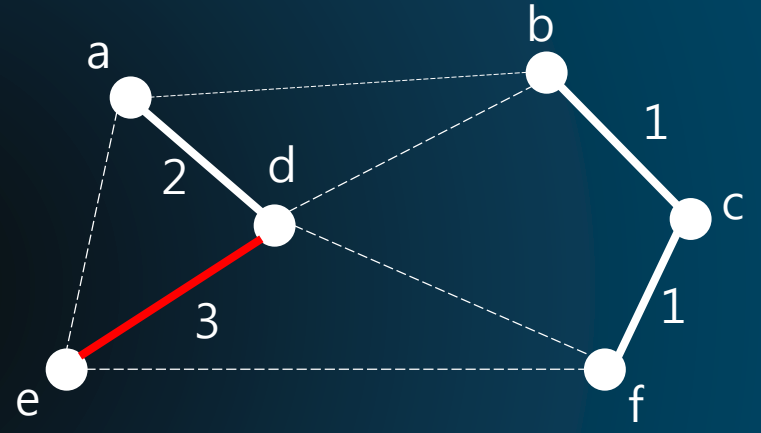


선분 (a,d) 추가

Kruskal 알고리즘의 실행과정(6)

정렬된 리스트 L

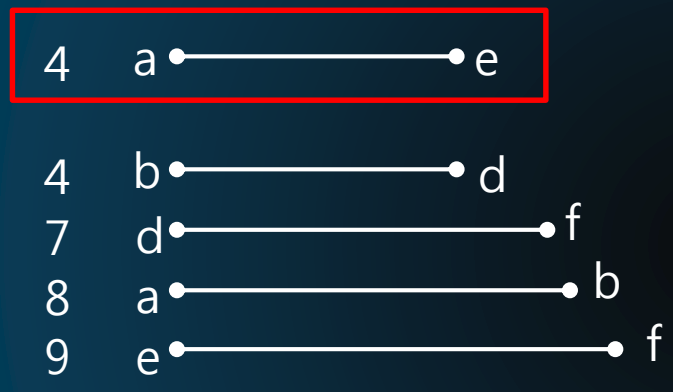


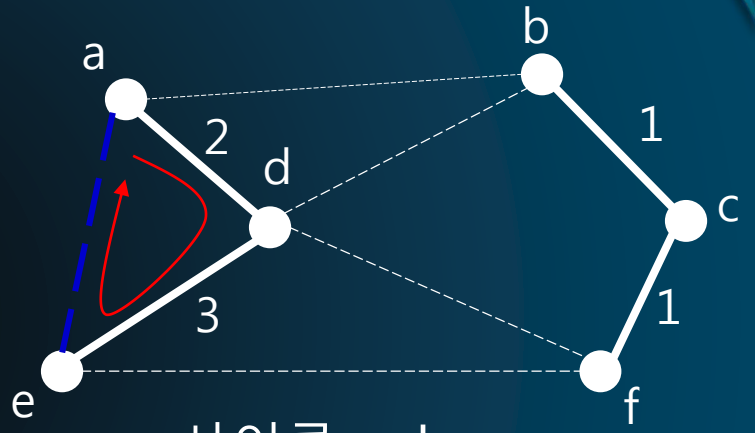


선분 (d,e) 추가

Kruskal 알고리즘의 실행과정(7)

정렬된 리스트 L



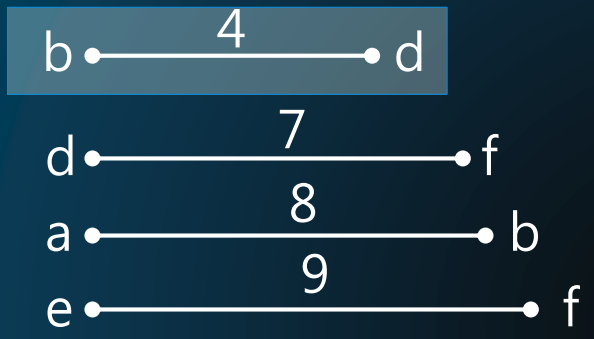


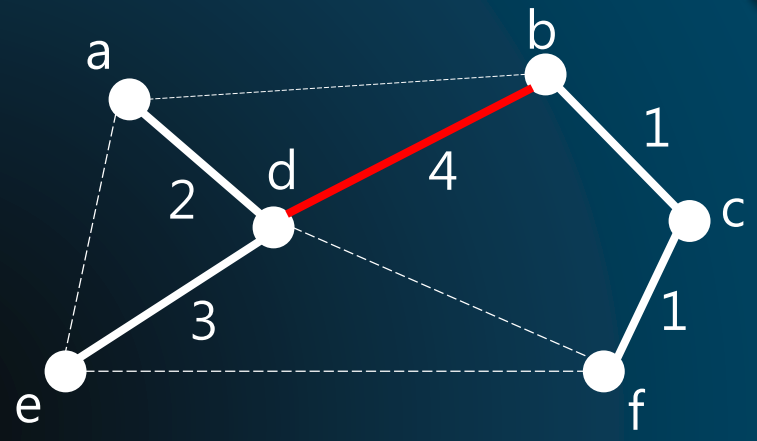
사이클 a-d-e-a

선분 (a,e) 버림

Kruskal 알고리즘의 실행과정(8)

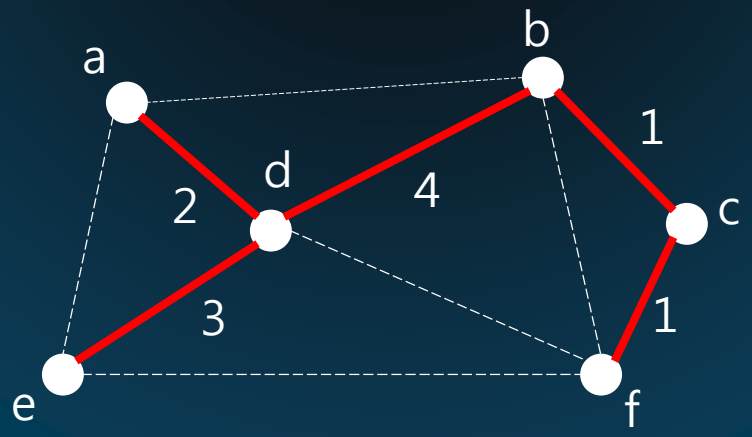
리스트
L





선분 (b,d) 추가

최종해



시간복잡도

- Line 1: 정렬하는데 O(mlogm) 시간. 단, m은 선분의 수이다.
- Line 2: T를 초기화하는 것이므로 O(1) 시간
- Line 3~8의 while-루프는 최악의 경우 m번 수행된다.
  즉, 그래프의 모든 선분이 while-루프 내에서 처리되는 경우이다.
  그리고 while-루프 내에서는 L로부터 가져온 선분 e가 사이클을
  만드는지를 검사하는데, 이는 $O(\log^* m)$ 시간이 걸린다.
- 따라서 크러스컬 알고리즘의 시간복잡도는 $O(m\log m)+O(m\log^* m)$
  $= O(m\log m)$이다.

❖ 반복로그 표현식, $O(\log^* n) = \begin{cases} 0 & , n \le 1 \\ \log^*(\log n), & n > 1 \end{cases}$

❖ 사이클 검사루틴: 상호배타집합(disjoint set)연산을 활용한 union과 find연산에 의해 트리구조에서의 사이클 존재유무를 체크하는 루틴.(연습문제 5번)
시간복잡도→ $O(\log^* m)$, m의 증가에 따라 매우 느리게 증가하는 함수.

BREAK TIME

Prim의 최소 신장 트리 알고리즘

- 주어진 가중치 그래프에서 임의의 점 하나를 선택한 후,

  (n-1)개의 선분을 하나씩 추가시켜서 트리를 확장해 나간다.

- 추가되는 선분은 현재까지 만들어진 트리에 연결시킬 때

  '욕심을 내어서' 항상 최소의 가중치로 연결되는 선분을 취한다.

Prim의 최소 신장 트리 알고리즘

PrimMST(G)

입력: 가중치 그래프 G=(V,E), |V|=n, |E|=m

출력: 최소 신장 트리 T

1. 그래프 G에서 임의의 점 p를 시작점으로 선택하고, D[p]=0으로 놓는다.

   // 배열 D[v]는 점 v에 대하여, T에 속한 점과 연결되어 있는 선분의 최소 가중치를 저장해두는 배열 D의 원소이다.

2. for (점 p가 아닌 각 점 v에 대하여) { // 배열 D의 초기화 과정(2~6)
3. if (선분 (p,v)가 그래프에 있으면)
4. D[v] = 선분 (p,v)의 가중치
5. else
6. D[v]=∞

   }

Prim의 최소 신장 트리 알고리즘(2)

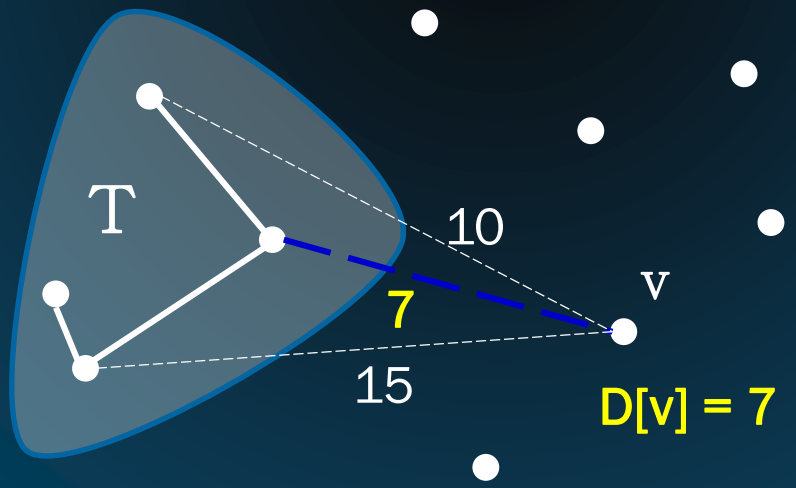
7. T= {p} // 초기에 트리 T는 점 p만을 가진다.

8. while (T에 있는 점의 수 < n) {
9. T에 속하지 않은 각 점 v에 대하여,
    D[v]가 최소인 점 v_{min}과 연결된 선분 (u,v_{min})을 T에 추가한다.
    단, u는 T에 속한 점이고, 점 v_{min}은 T에 새로 추가된다.
10. for (T에 속하지 않은 각 점 w에 대해서) { // D[w]를 갱신 과정
11. if (선분 (v_{min},w)의 가중치 < D[w]) // 새로 추가된 점 v_{min}과 비교하여
12. D[w] = 선분 (v_{min},w)의 가중치 // 기존 값보다 작으면 갱신한다.
    }
  }

13. return T // T는 최소 신장 트리이다.

PrimMST 알고리즘의 설명(생략)

- Line 1: 임의로 점 p를 선택하고, D[p]=0으로 놓는다. 여기서 배열 D[v]에는 점 v와 T에 속한 점들을 연결하는 선분들 중에서 최소 가중치를 가진 선분의 가중치를 저장한다.
  - 다음그림에서 D[v]에는 10, 7, 15 중에서 최소 가중치인 7이 저장된다.



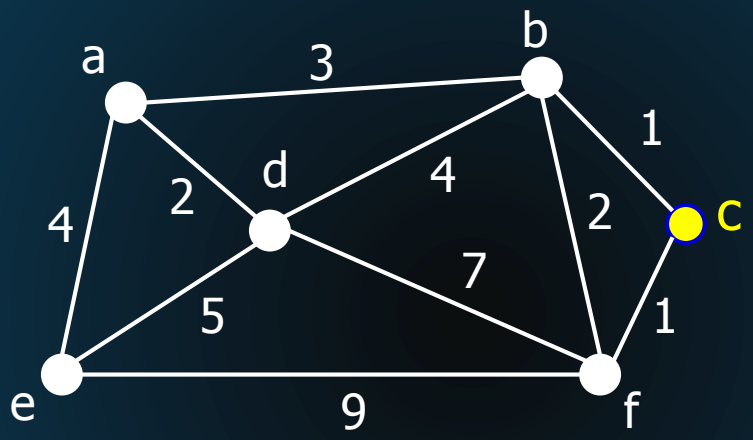
PrimMST 알고리즘의 설명(2)

- Line 2~6: 시작점 p와 선분으로 연결된 점 v의 D[v]를 선분 (p,v)의 가중치로 초기화시키고, 점 p와 선분으로 연결되지 않은 점 v에 대해서 D[v]=∞로 놓는다.
- Line 7: T = {p}로 초기화시킨다.
- Line 8~12의 while-루프는 T의 점의 수가 n이 될 때까지 수행된다. T에 속한 점의 수가 n이 되면, T는 신장 트리이다.
- Line 9: T에 속하지 않은 각 점 v에 대하여, D[v]가 최소인 점 v_{min}을 찾는다. 그리고 점 v_{min}과 연결된 선분 (u,v_{min})을 T에 추가한다. 단, u는 T에 속한 점이고, 선분 (u,v_{min})이 T에 추가된다는 것은 점 v_{min}도 T에 추가되는 것이다.

PrimMST 알고리즘의 설명(3)

- Line 10~12의 for-루프에서는 line 9에서 새로이 추가된 점 v_{min}에 연결되어 있으면서 T에 속하지 않은 각 점 w의 D[w]를 선분 (v_{min},w)의 가중치가 D[w]보다 작으면 (if-조건), D[w]를 선분 (v_{min},w)의 가중치로 갱신한다.
- 마지막으로 line 13에서는 최소 신장 트리 T를 리턴한다.

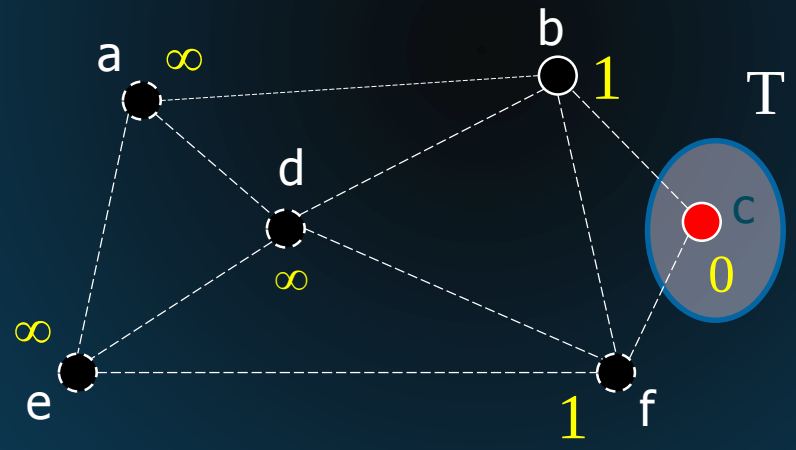
PrimMST 알고리즘의 실행 과정



- Line 1: 임의의 시작점으로 점 c가 선택되었다고 가정하자.
  그리고 D[c]=0으로 초기화시킨다.

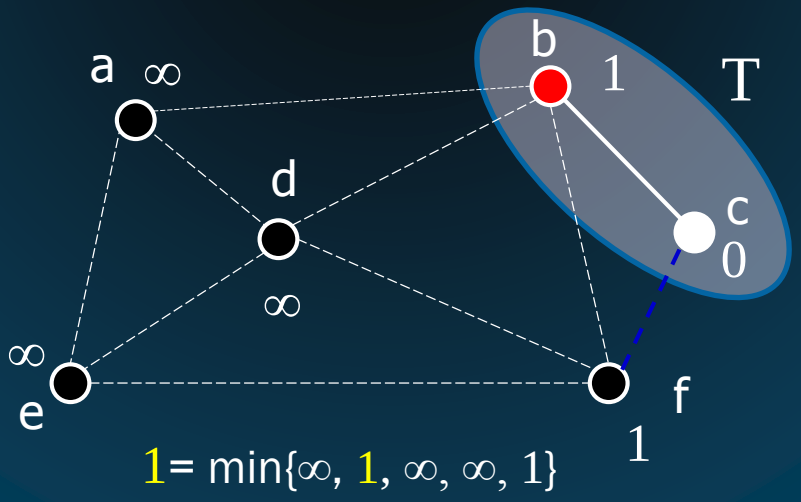
PrimMST 알고리즘의 실행 과정(2)

- Line 2~6: 시작점 c와 선분으로 연결된 각 점 v에 대해서,
  D[v]를 각 선분의 가중치로 초기화시키고,
  나머지 각 점 w에 대해서, D[w]는 ∞로 초기화시킨다.



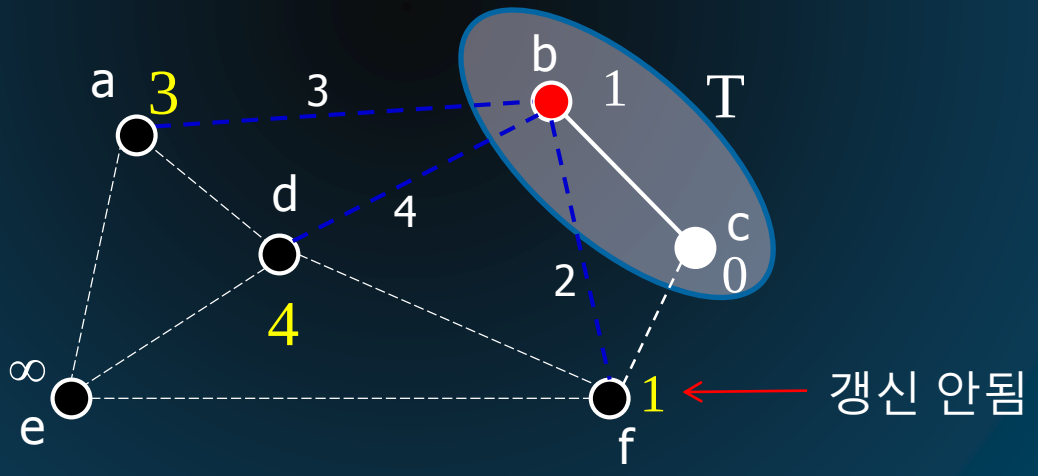
PrimMST 알고리즘의 실행 과정(3)

- Line 7: T={c}로 초기화한다.
- Line 8의 while-루프의 조건이 '참'이다. 즉, 현재 T에는 점 c만이 있다.
  따라서 line 9에서 T에 속하지 않은 각 점 v에 대하여,
  D[v]가 최소인 점 v_{min}을 선택한다. D[b]=D[f]=1로서 최소값이므로
  점 b나 점 f 중 택일. 여기서는 점 b를 선택하자.
  따라서 점 b와 선분 (c,b)가 T에 추가된다.



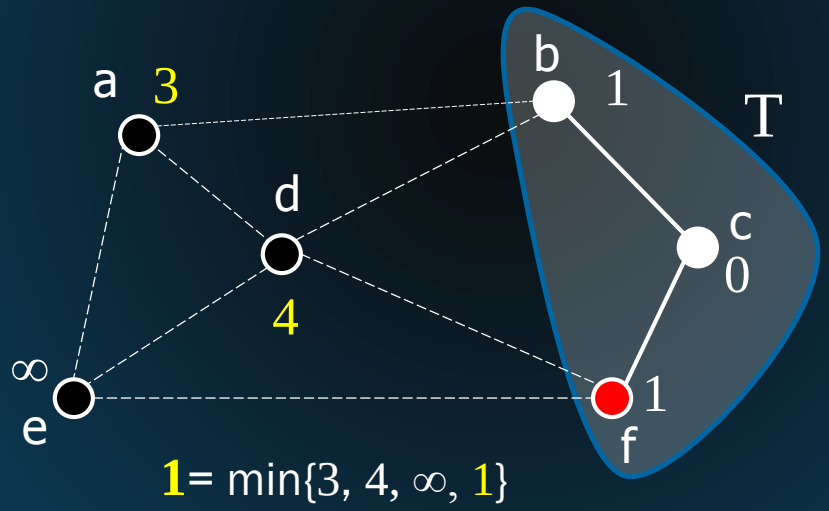
PrimMST 알고리즘의 실행 과정(4)

- Line 10~12: 점 b에 연결된 점 a와 d의 D[a]와 D[b]를 각각 3과 4로 갱신한다. 점 f는 점 b와 선분으로 연결은 되어있으나,
  선분 (b,f)의 가중치인 2가 현재 D[f]보다 크므로 D[f]를 갱신 안됨



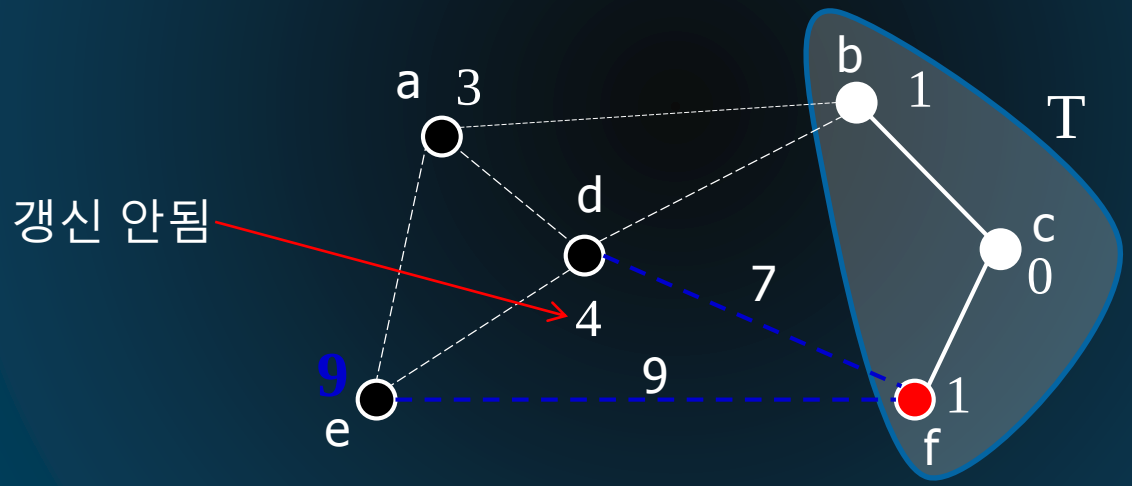
PrimMST 알고리즘의 실행 과정(5)

- Line 8의 while-루프의 조건이 '참'이므로,
  line 9에서 T에 속하지 않은 각 점 v에 대하여,
  v_{min}인 점 f를 찾고, 점 f와 선분 (c,f)를 T에 추가시킨다.



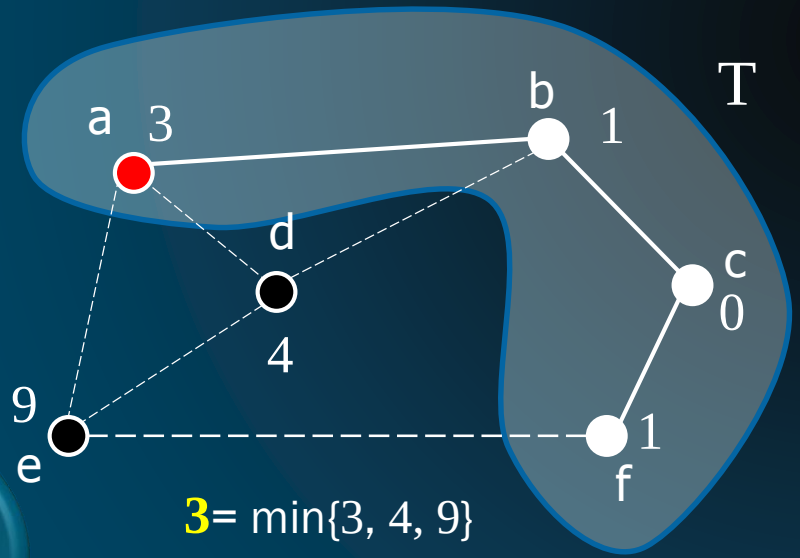
PrimMST 알고리즘의 실행 과정(6)

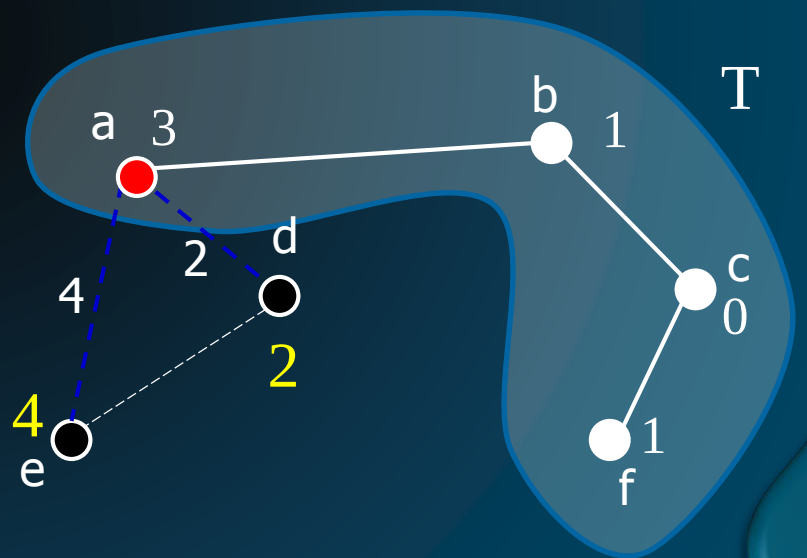
- Line 10~12: 점 f에 연결된 점 e의 D[e]를 9로 갱신한다.
  D[d]는 선분 (f,d)의 가중치인 7보다 작기 때문에 갱신안됨



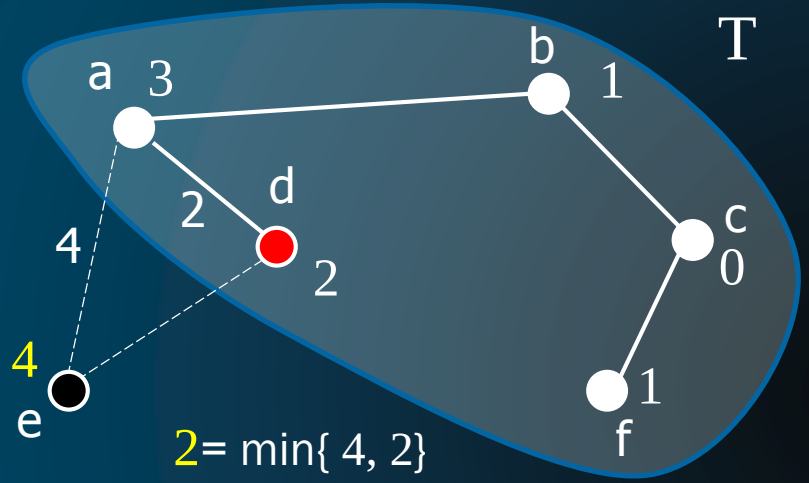
PrimMST 알고리즘의 실행 과정(7)

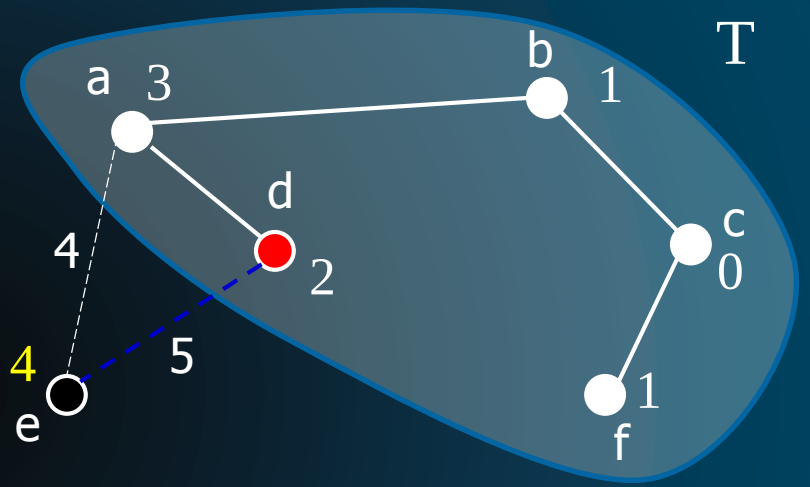
- 그 다음부터는 점 a와 선분 (b,a), 점 d와 선분 (a,d)가 차례로 T에 추가되고, 최종적으로 점 e와 선분 (a,e)가 추가되면서, 최소 신장 트리 T가 완성된다. Line 13에서는 T를 리턴하고, 알고리즘을 마친다.

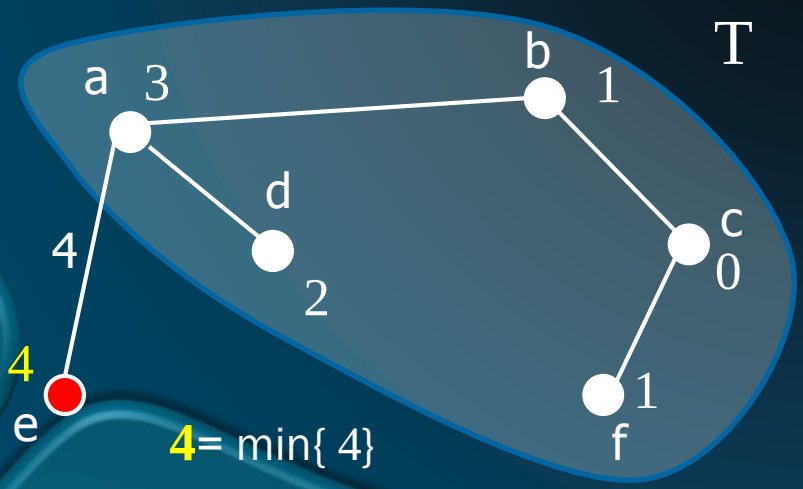


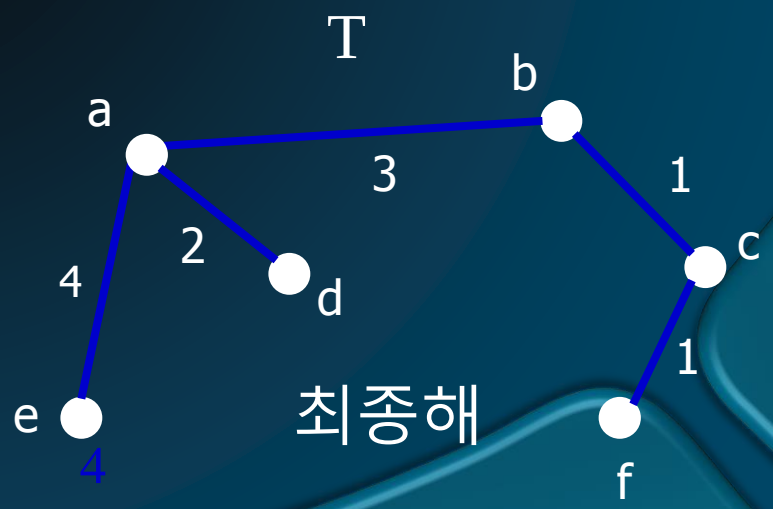


PrimMST 알고리즘의 실행 과정(8)



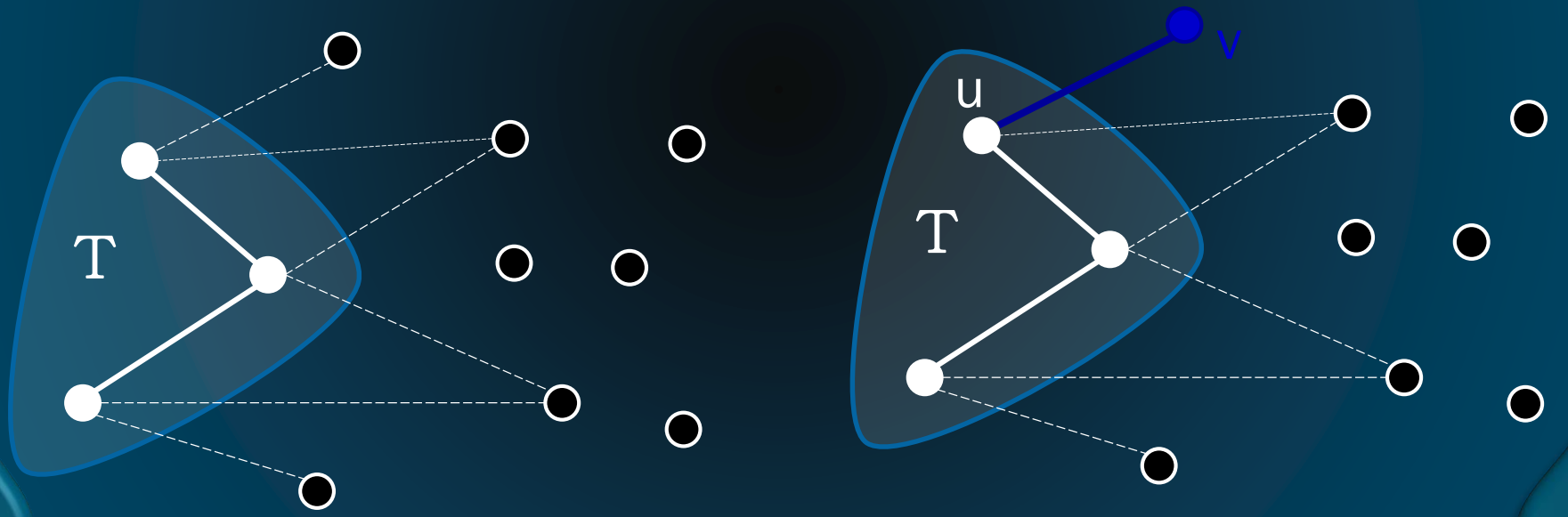






PrimMST 알고리즘에서 싸이클 생성불가

- PrimMST 알고리즘이 최종적으로 리턴하는 T에는 왜 사이클이 없을까?
  - 프림 알고리즘은 T 밖에 있는 점을 항상 추가하므로 사이클이 만들어지지 않는다.



시간복잡도

- while-루프가 (n-1)번 반복되고,
  1회 반복될 때 line 9에서 T에 속하지 않은 각 점 v에 대하여,
  D[v]가 최소인 점 v_{min}을 찾는데 O(n) 시간이 걸린다.
  - 왜냐하면 1차원 배열 D에서 (현재 T에 속하지 않은 점들에 대해서)
    최소값을 찾는 것이고, 배열의 크기는 그래프의 점의 수인 n이기 때문이다.
- 프림 알고리즘의 시간복잡도: $(n-1)xO(n) = O(n^2)$

Kruskal과 Prim의 수행과정 비교

- 크러스컬 알고리즘에서는 선분이 1개씩 T에 추가되는데,
  이는 마치 n개의 점들이 각각의 트리인 상태에서 선분이 추가되면
  2개의 트리가 1개의 트리로 합쳐지는 것과 같다.
  크러스컬 알고리즘은 이를 반복하여 1개의 트리인 T를 만든다.
  즉, n개의 트리들이 점차 합쳐지면서 1개의 신장 트리가 만들어진다.

- 프림 알고리즘에서는 T가 점 1개인 트리에서 시작되어
  선분을 1개씩 추가시킨다.
  즉, 1개의 트리가 자라나서 신장 트리가 된다.

응 용 분 야

- 최소 비용으로 선로 또는 파이프 네트워크를 설치하는데 활용
  - 인터넷 광 케이블 선로,
  - 케이블 TV선로,
  - 전화선로,
  - 송유관로,
  - 가스관로,
  - 배수로 등
- 여행자 문제(Traveling Salesman Problem)를 근사적으로 해결하는데 이용

4.3 최단 경로 찾기

- 단일 출발점 최단 경로 (Single Source Shortest Path) 문제는 주어진 가중치 그래프에서 출발점(Single Source)에서 시작하여 다른 모든 도착점까지의 최단 경로를 찾는 문제이다.
- 최단 경로를 찾는 대표적인 알고리즘은 다익스트라(Dijkstra) 최단 경로 알고리즘이며, 이 또한 그리디 알고리즘이다.

Edsger W. Dijkstra(1930-2002)

- Dutch computer scientist (1930 – 2002)
- “Structured Programming”
- “Go To Statement Considered Harmful” (1968)
- Turing award (1972)

Dijkstra’s algorithm for
single source shortest paths problem
(mid-1950s)

기본 아이디어

- 프림MST 알고리즘과 거의 흡사한 과정으로 진행된다.
- 2가지 차이점:

1. 프림 알고리즘은 임의의 점에서 시작하나,
   다익스트라 알고리즘은 주어진 출발점에서 시작한다.

2. 프림 알고리즘은 트리에 하나의 점(선분)을 추가시킬 때
   기존 트리에서 가장 가까운 점을 추가시킨다.
   다익스트라 알고리즘은 출발점으로부터 최단 거리가 확정되지 않은 점들 중에서 출발점으로부터 가장 가까운 점을 추가하고,
   그 점의 최단 거리를 확정한다.

다익스트라 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

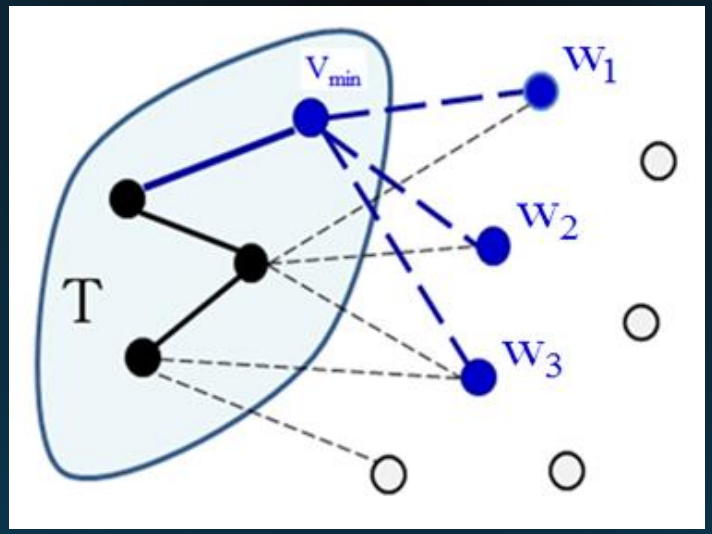
1. 배열 D를 ∞로 초기화시킨다. 단, D[s]=0으로 초기화한다.
   // 배열 D[v]에는 출발점 s로부터 점 v까지의 거리가 저장된다.
2. while (s로부터의 최단 거리가 확정되지 않은 점이 있으면) {
3. 현재까지 s로부터 최단 거리가 확정되지 않은 각 점 v에 대해서 최소의 D[v]의 값을 가진 점 v_{min}을 선택하고, 출발점 s로부터 점 v_{min}까지의 최단 거리 D[v_{min}]을 확정시킨다.
4. s로부터 현재보다 짧은 거리로 점 v_{min}을 통해 우회 가능한 각 점 w에 대해서 D[w]를 갱신한다.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

ShortestPath 알고리즘의 설명(생략)

- 알고리즘에서 배열 D[v]는 출발점 s로부터 점 v까지의 거리를 저장하는데 사용하고, 최종적으로는 출발점 s로부터 점 v까지의 최단 거리를 저장하게 된다. Line 1에서는 출발점 s의 D[s]=0으로, 또 다른 각 점 v에 대해서 D[v]=∞로 초기화시킨다.
- Line 2~4의 while-루프는 (n-1)회 수행된다. 현재까지 s로부터 최단 거리가 확정된 점들의 집합을 T라고 놓으면, V-T는 현재까지 s로부터 최단 거리가 확정되지 않은 점들의 집합이다. 따라서 V-T에 속한 각 점 v에 대해서 D[v]가 최소인 점 v_{min}을 선택하고, v_{min}의 최단 거리를 확정시킨다.
  즉, $D[v_{min}] \le D[v]$, $v \in V-T$이다. '확정한다는 것'은 2가지의 의미를 갖는다.
  - $D[v_{min}]$이 확정된 후에는 다시 변하지 않는다.
  - 점 v_{min}이 T에 포함된다

ShortestPath 알고리즘의 설명(2)

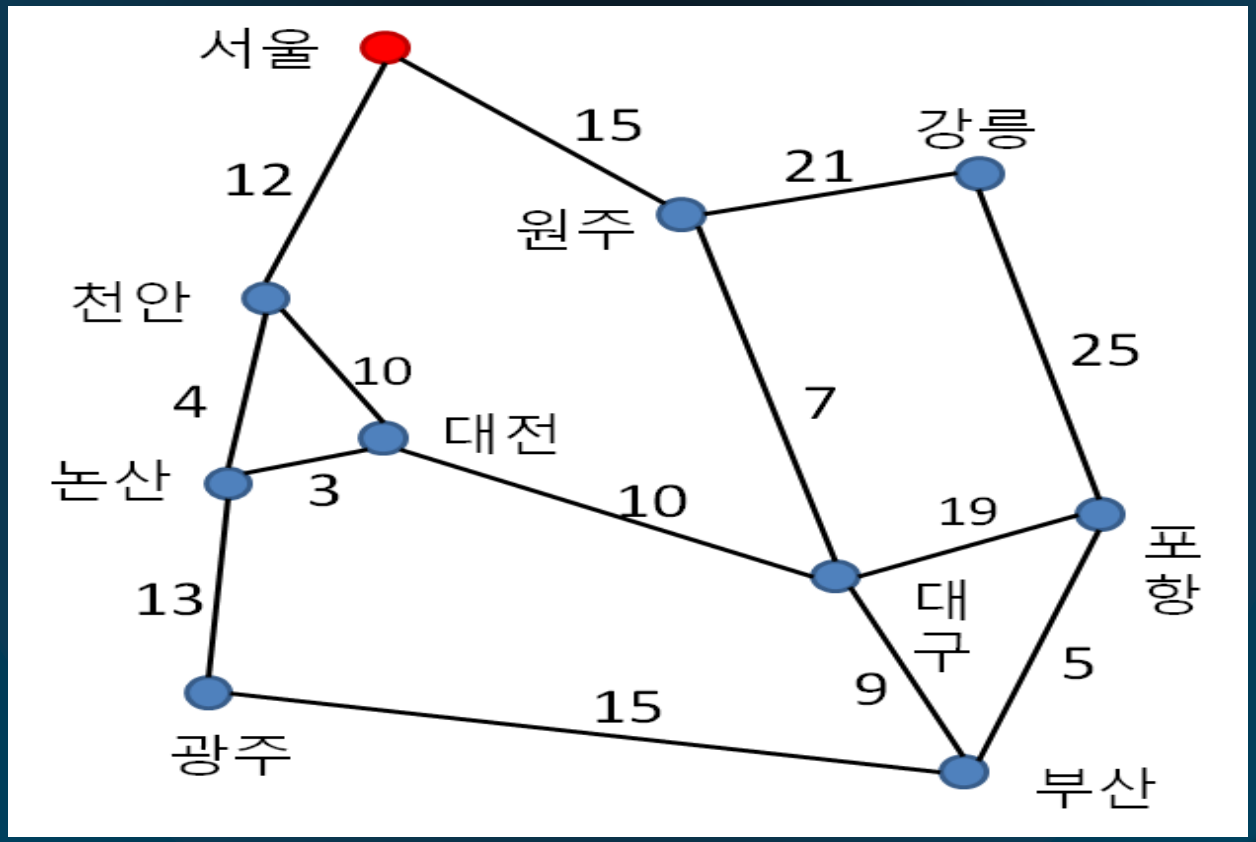
- Line 4: V-T에 속한 점들 중 v_{min}을 거쳐 감 (경유함)으로서 s로부터의 거리가 현재보다 더 짧아지는 점 w가 있으면, 그 점의 D[w]를 갱신한다.
- 다음 그림은 v_{min}이 T에 포함된 상태를 보이고 있는데, v_{min}에 인접한 점 w_1, w_2, w_3 각각에 대해서 만일 ($D[v_{min}]$+선분 (v,w_i)의 가중치)<$D[w_i]$이면, $D[w_i]$ = ($D[v_{min}]$+선분(v_{min},w_i)의 가중치)로 갱신한다.



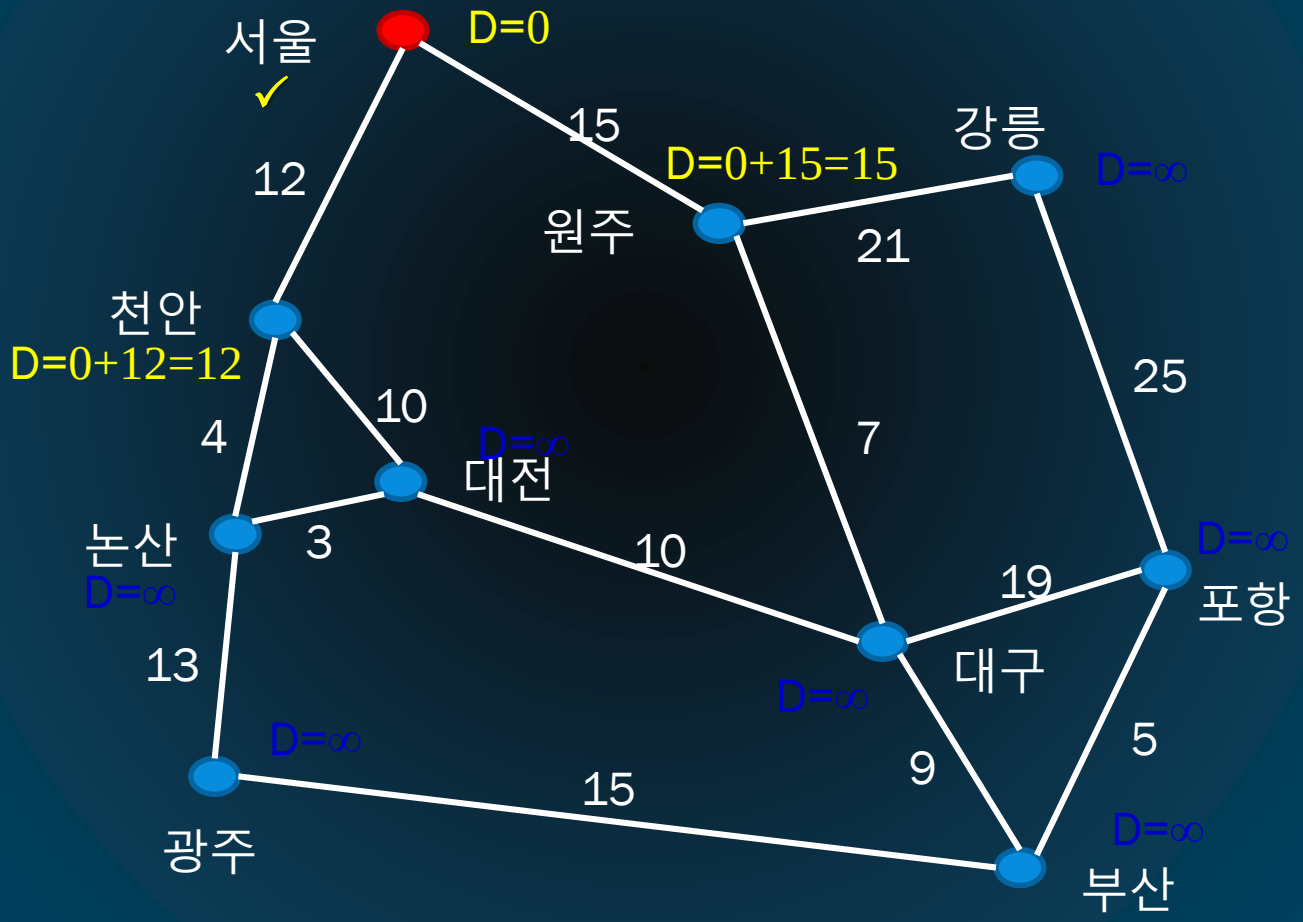
- 마지막으로 line 5에서 배열 D를 리턴한다.

ShortestPath 알고리즘의 수행 과정

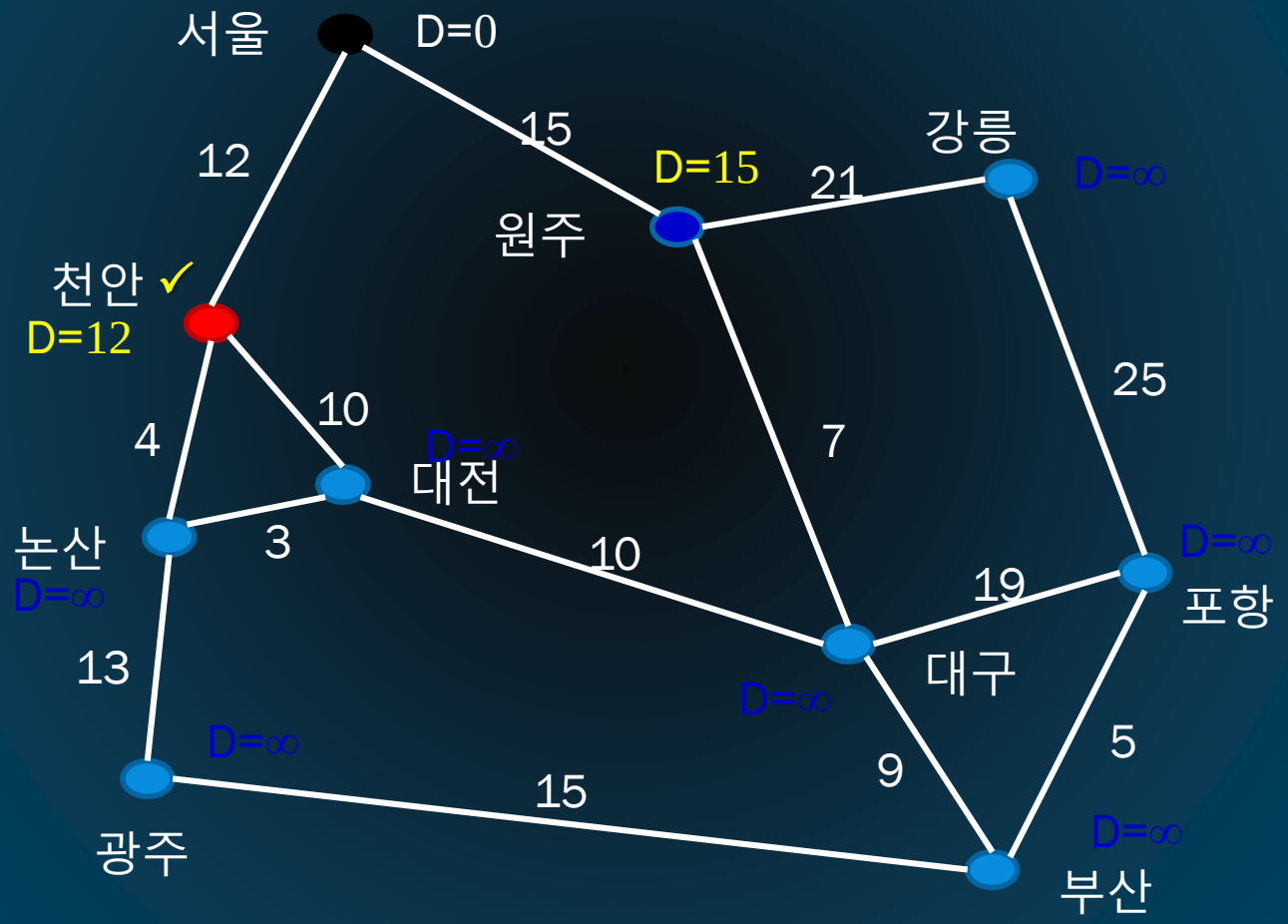
- 단, 출발점은 서울이다.



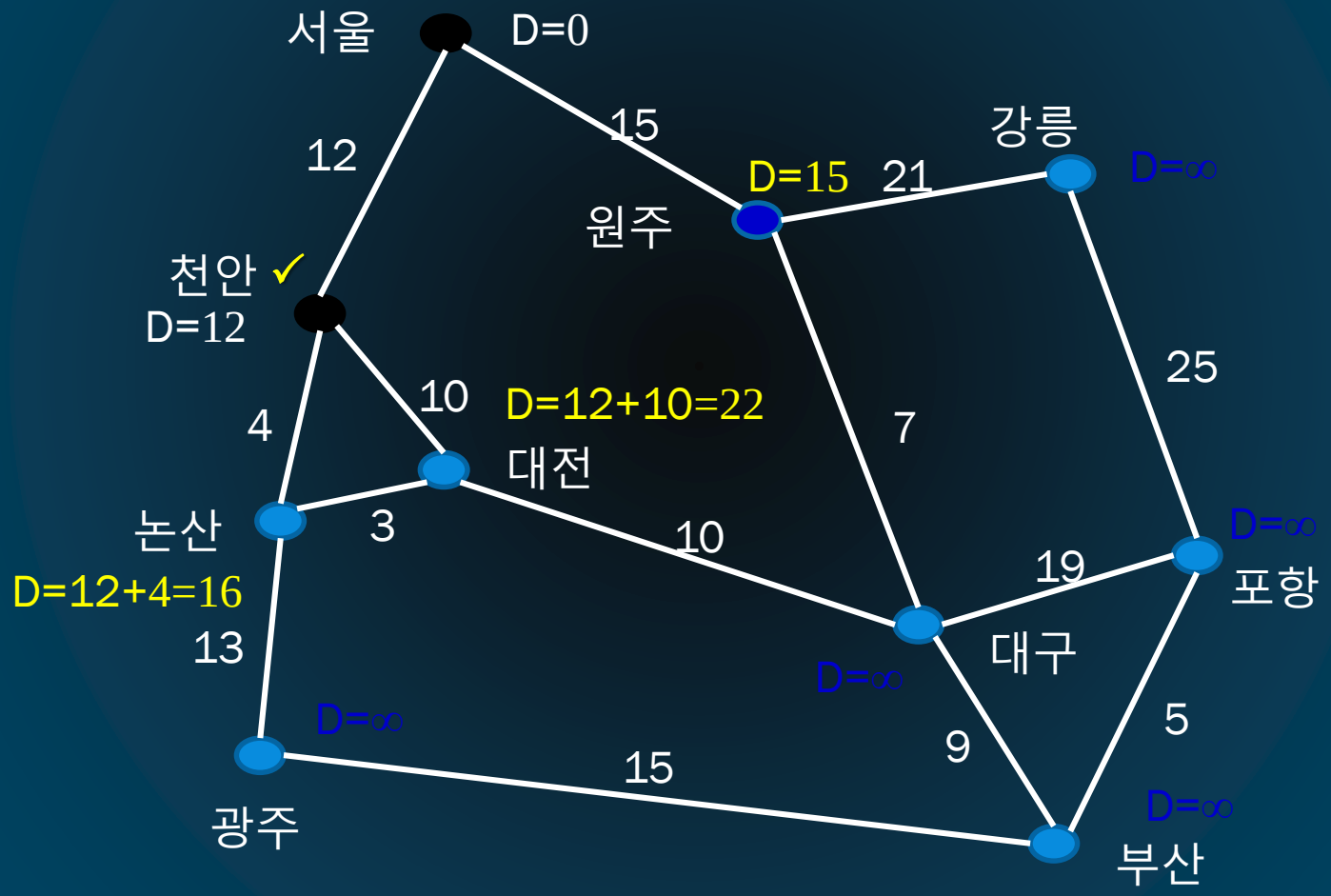
ShortestPath 알고리즘의 수행 과정(2)



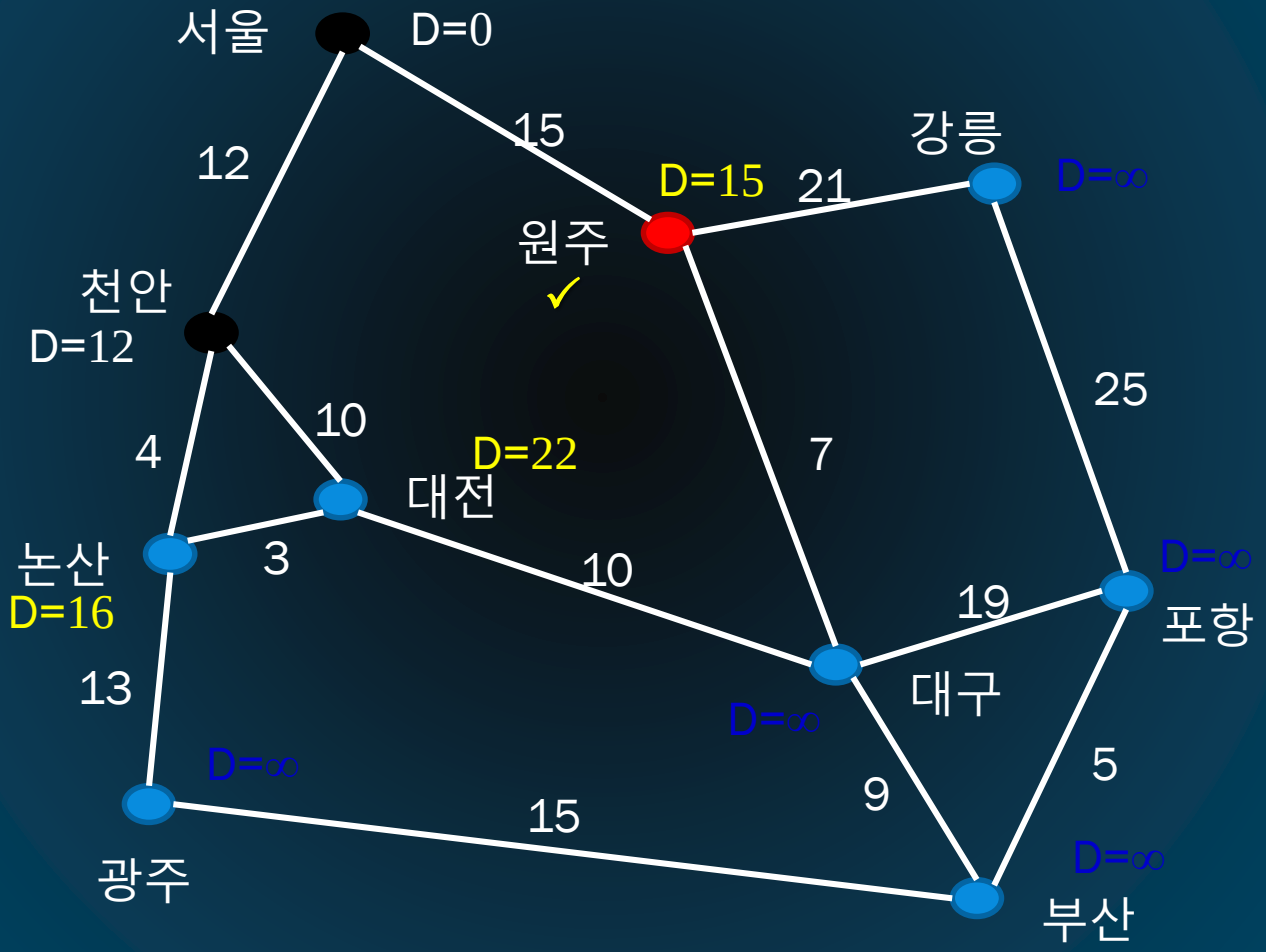
ShortestPath 알고리즘의 수행 과정(3)



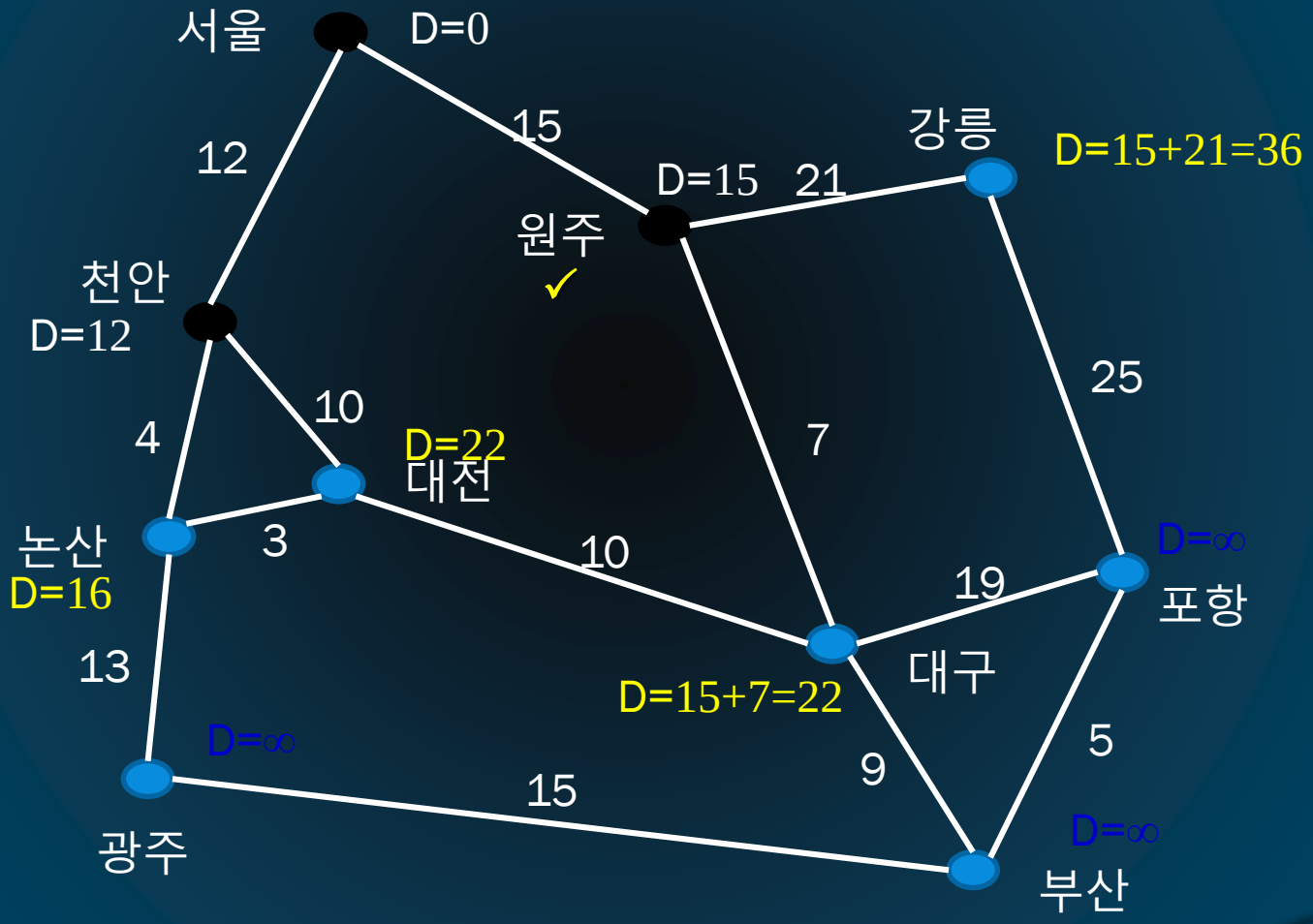
ShortestPath 알고리즘의 수행 과정(4)



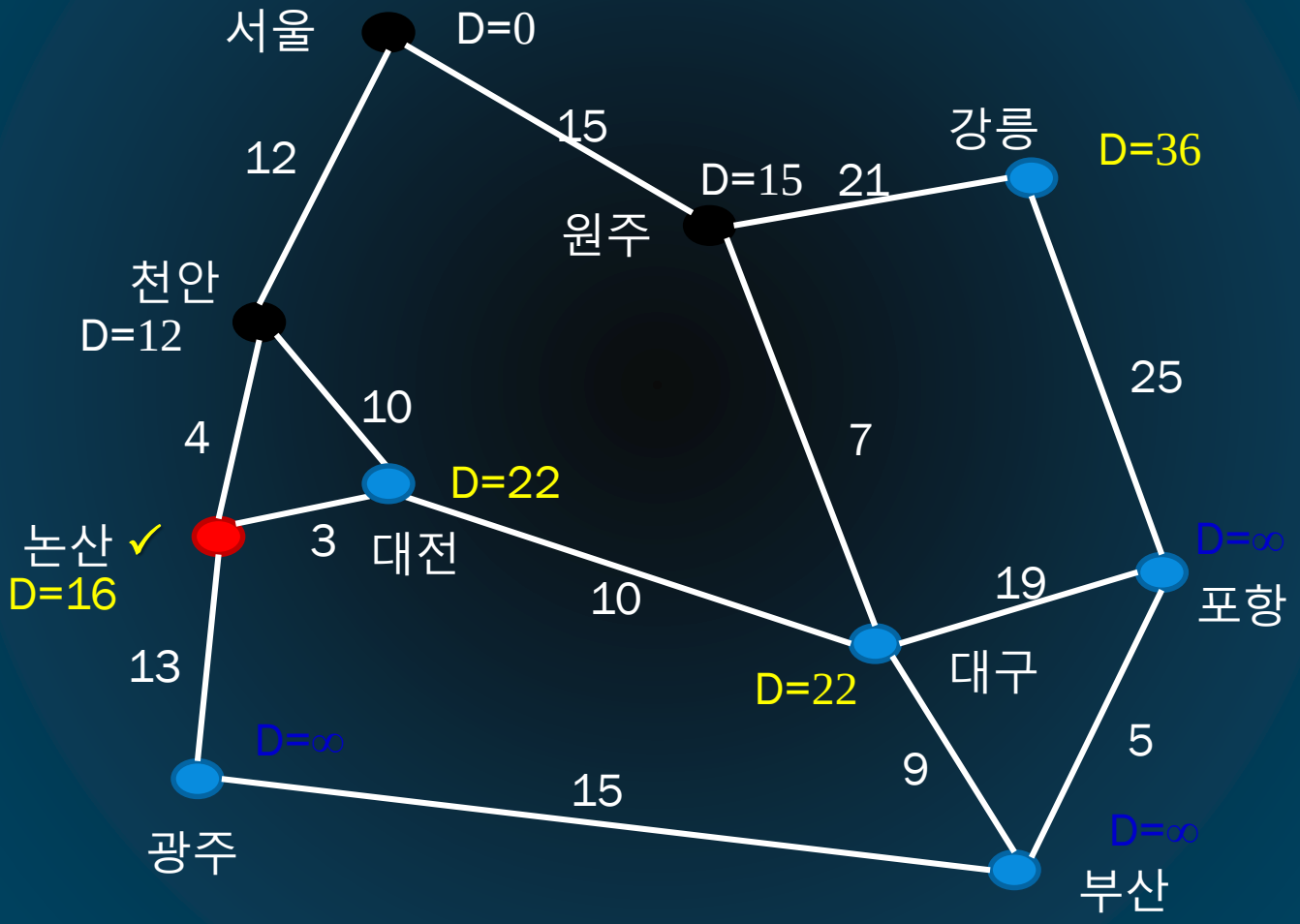
ShortestPath 알고리즘의 수행 과정(5)



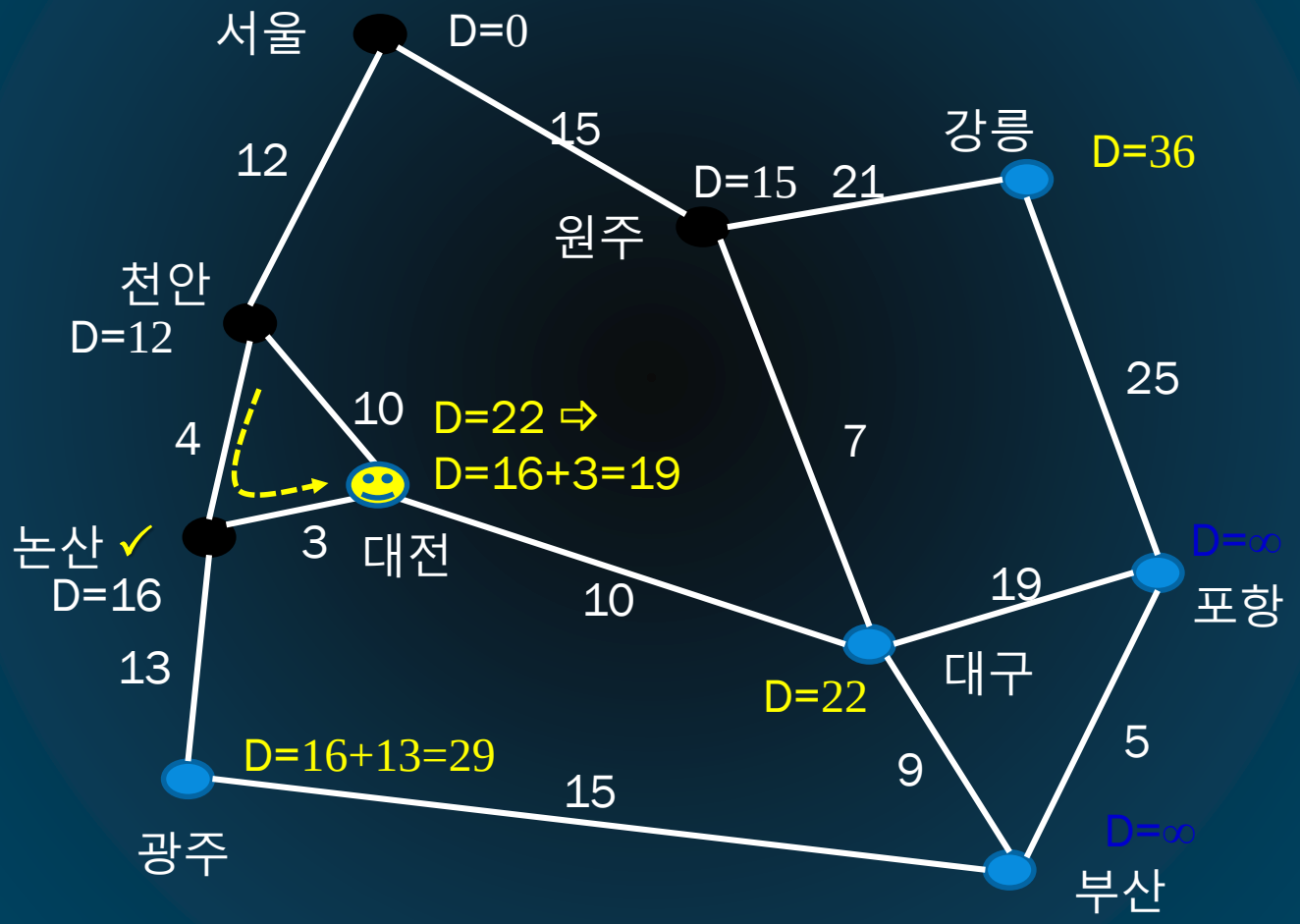
ShortestPath 알고리즘의 수행 과정(6)



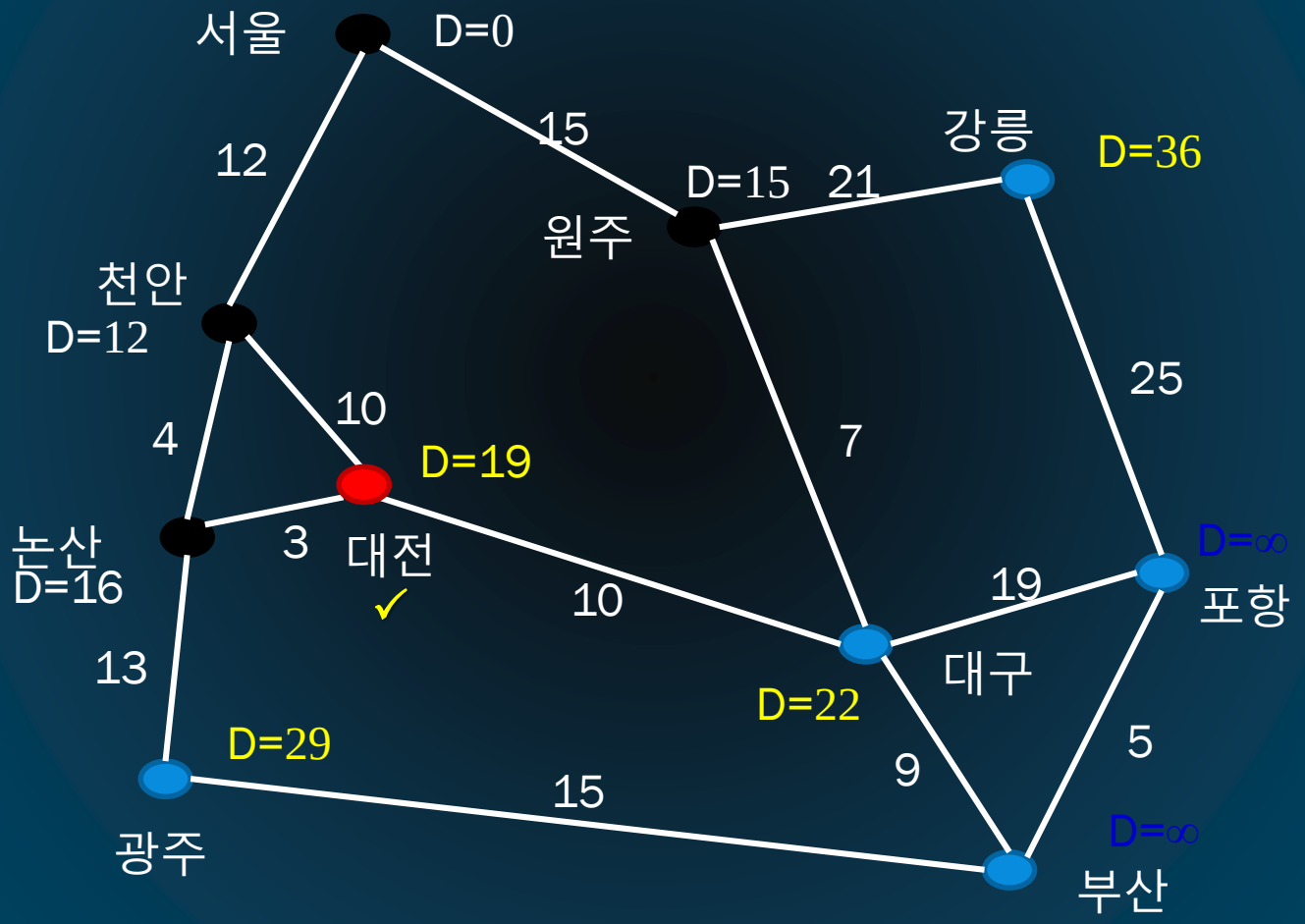
ShortestPath 알고리즘의 수행 과정(7)



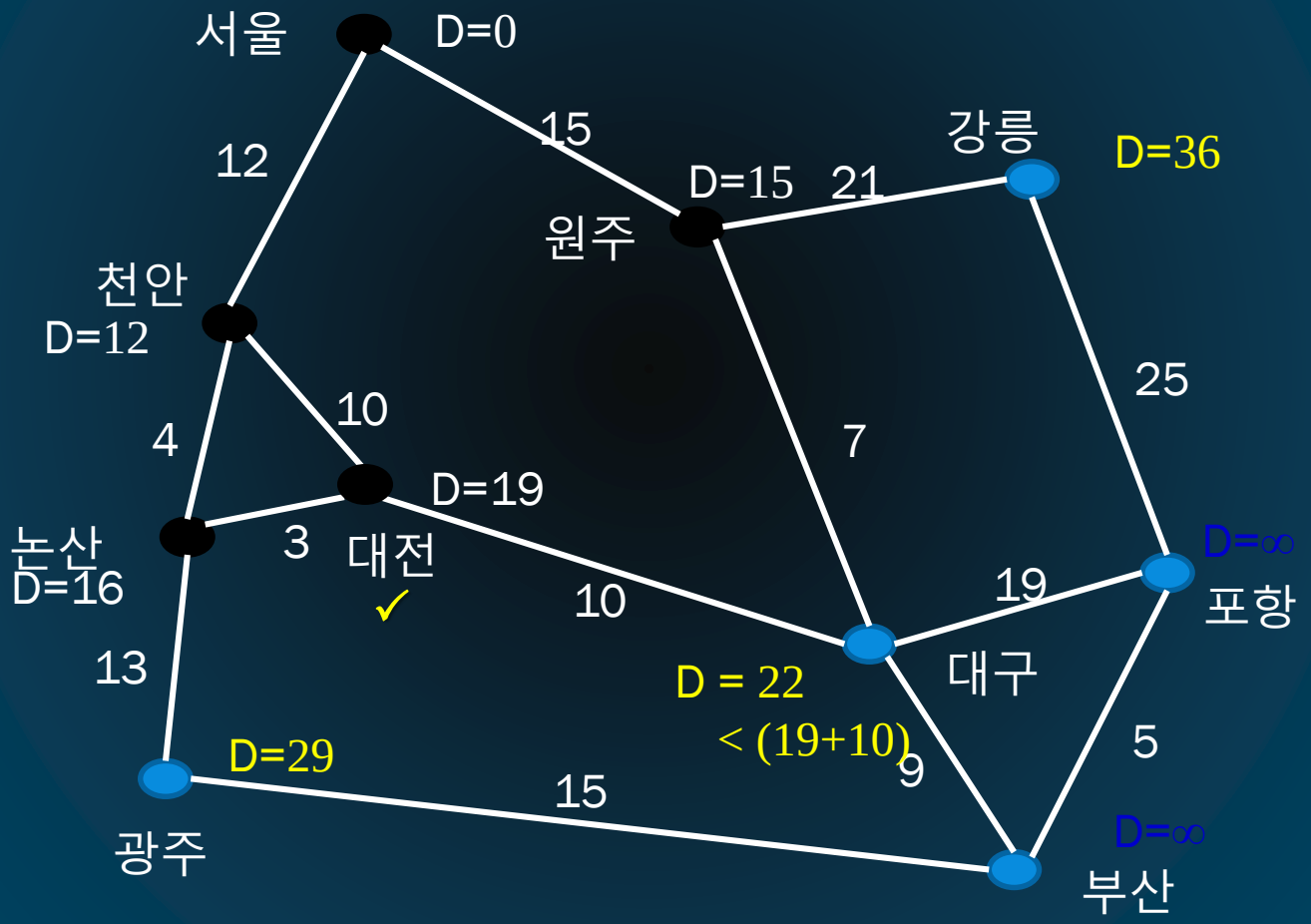
ShortestPath 알고리즘의 수행 과정(8)



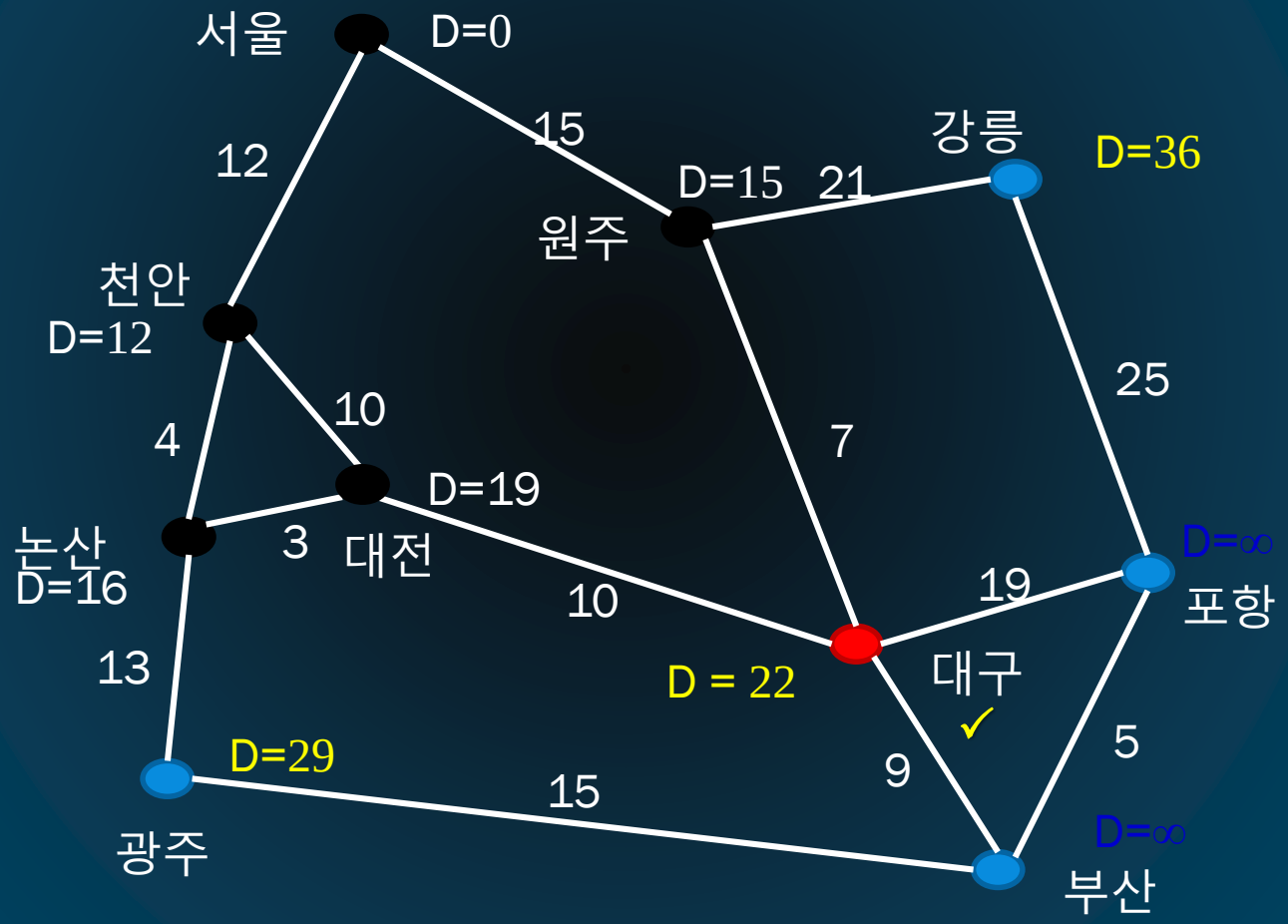
ShortestPath 알고리즘의 수행 과정(9)



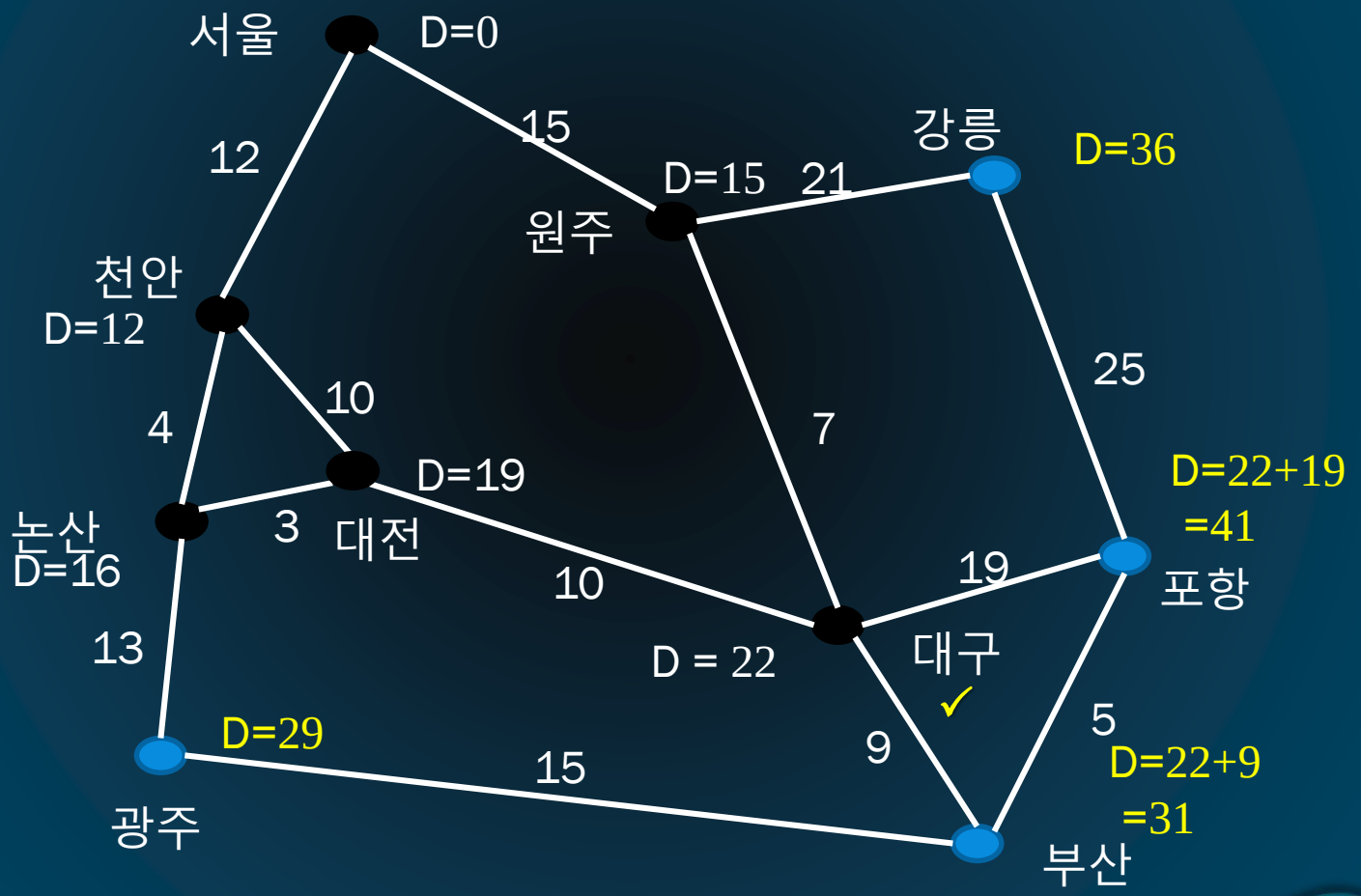
ShortestPath 알고리즘의 수행 과정(10)



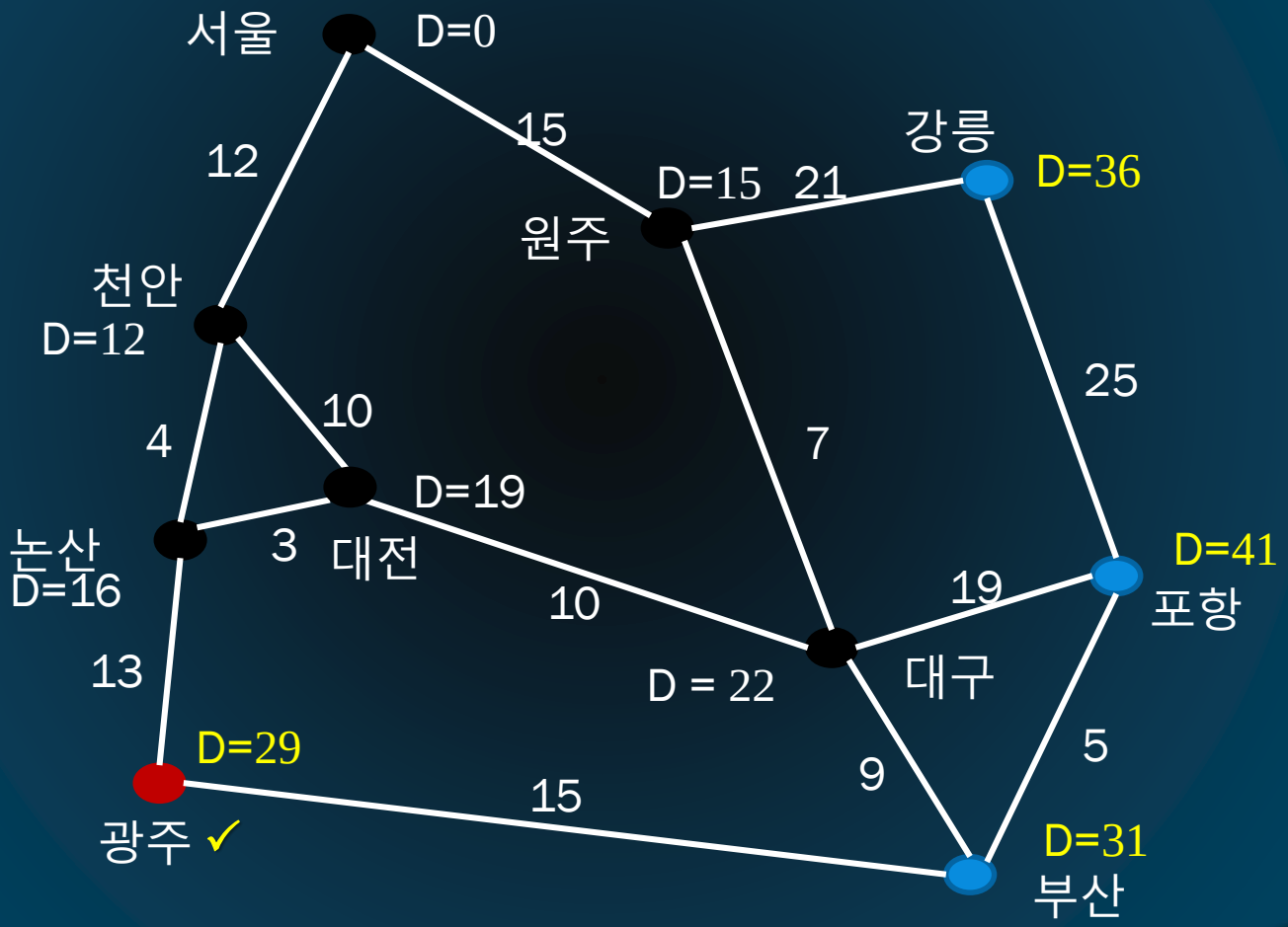
ShortestPath 알고리즘의 수행 과정(11)



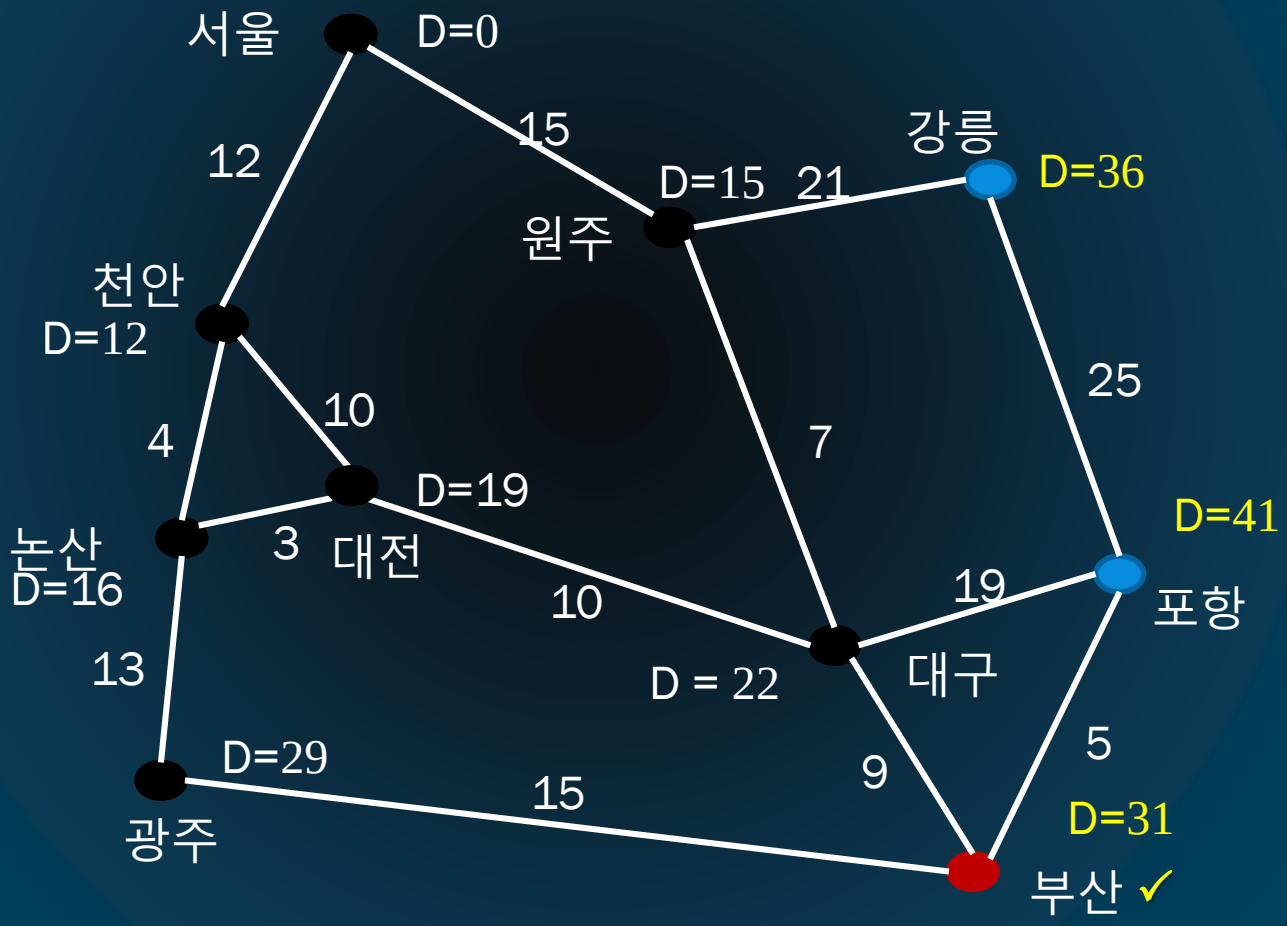
ShortestPath 알고리즘의 수행 과정(12)



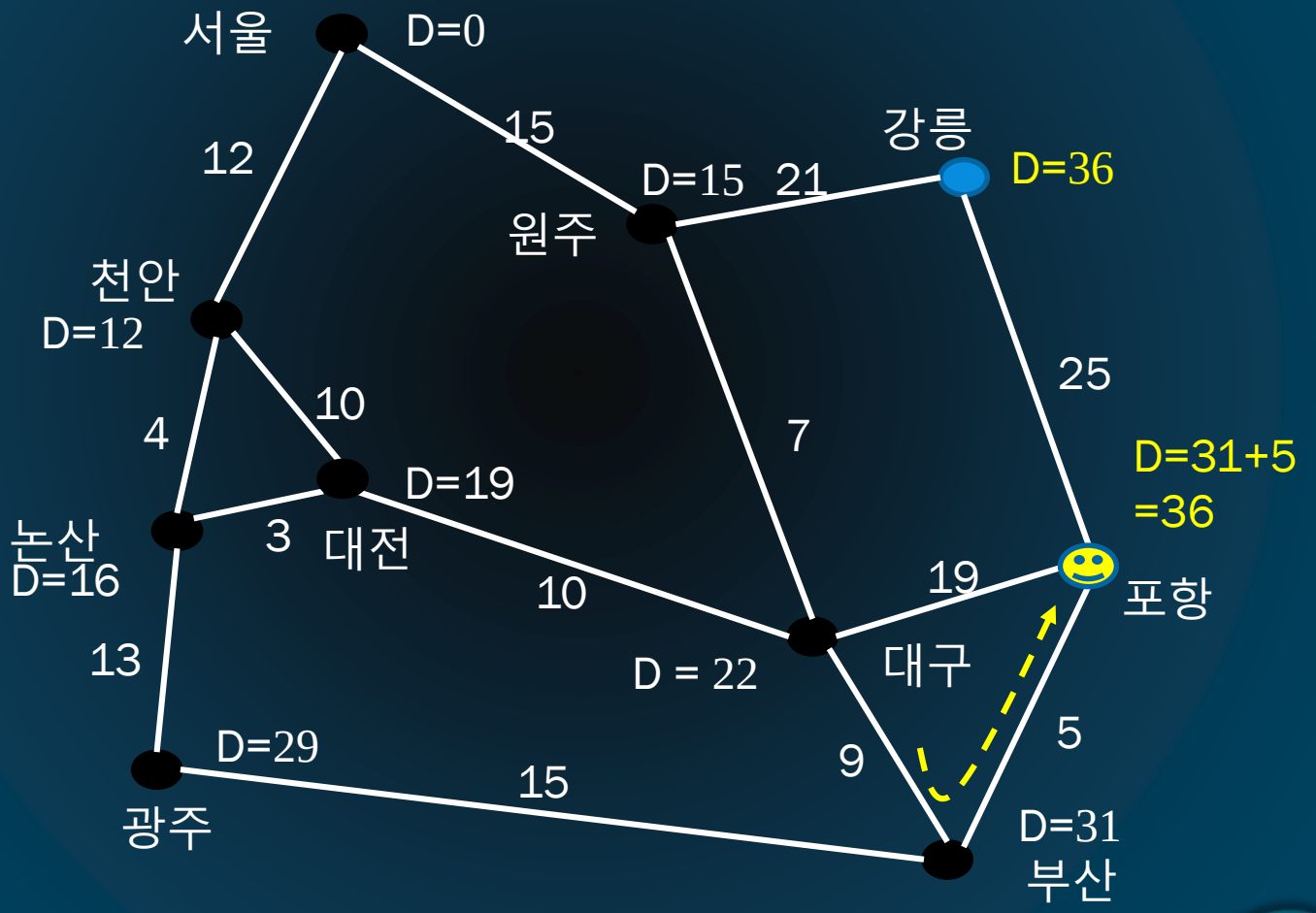
ShortestPath 알고리즘의 수행 과정(13)



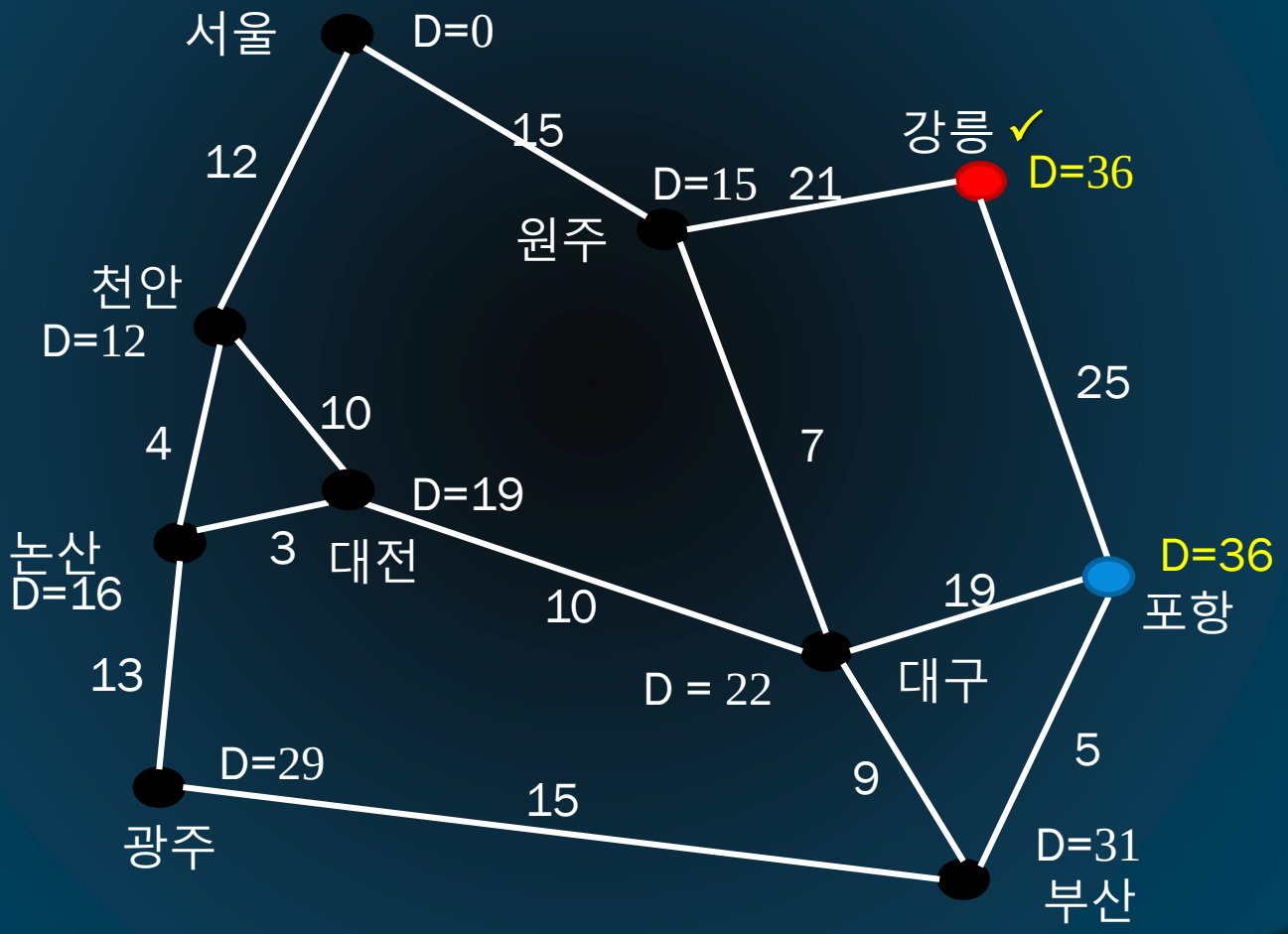
ShortestPath 알고리즘의 수행 과정(14)



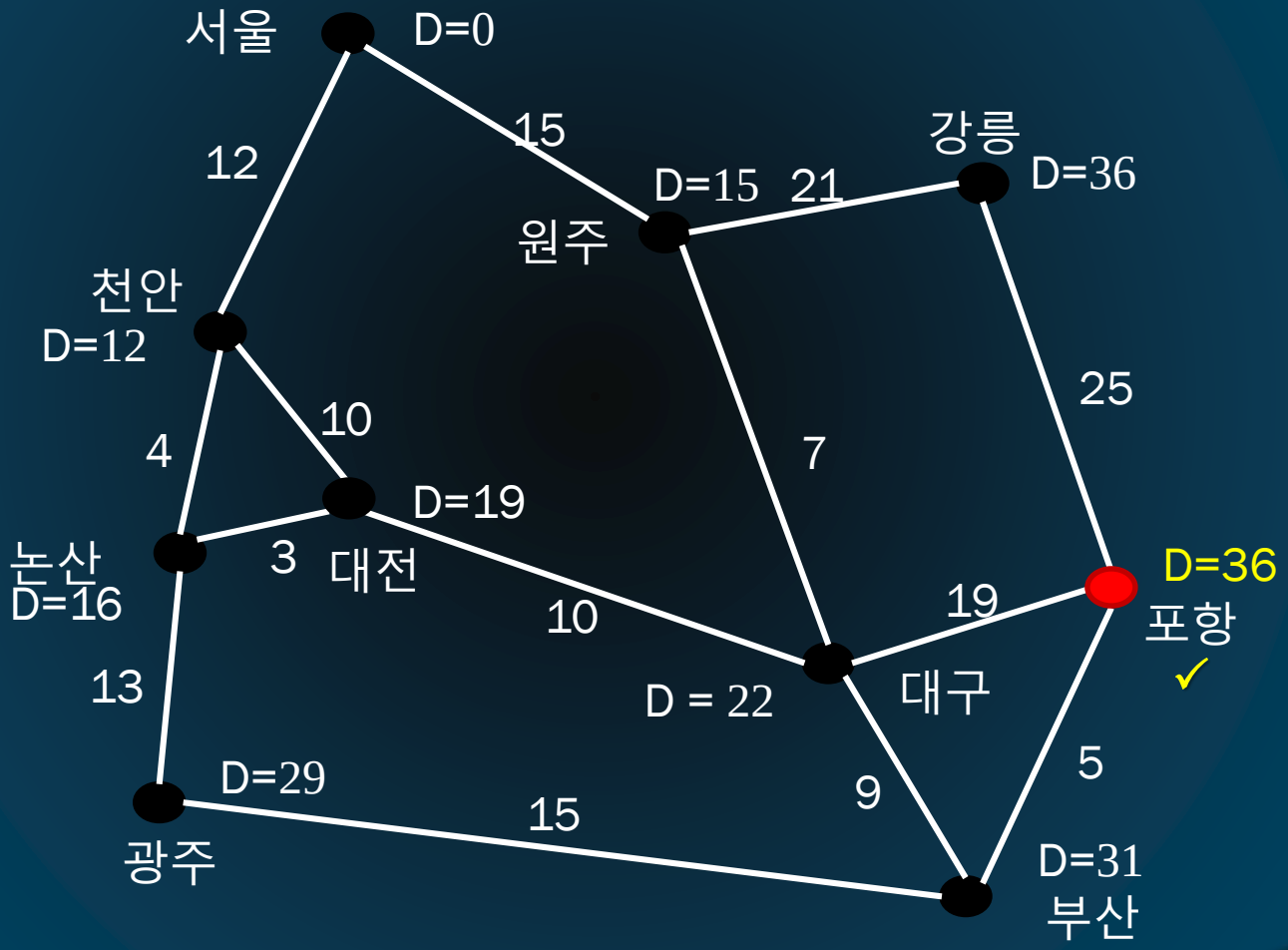
ShortestPath 알고리즘의 수행 과정(15)



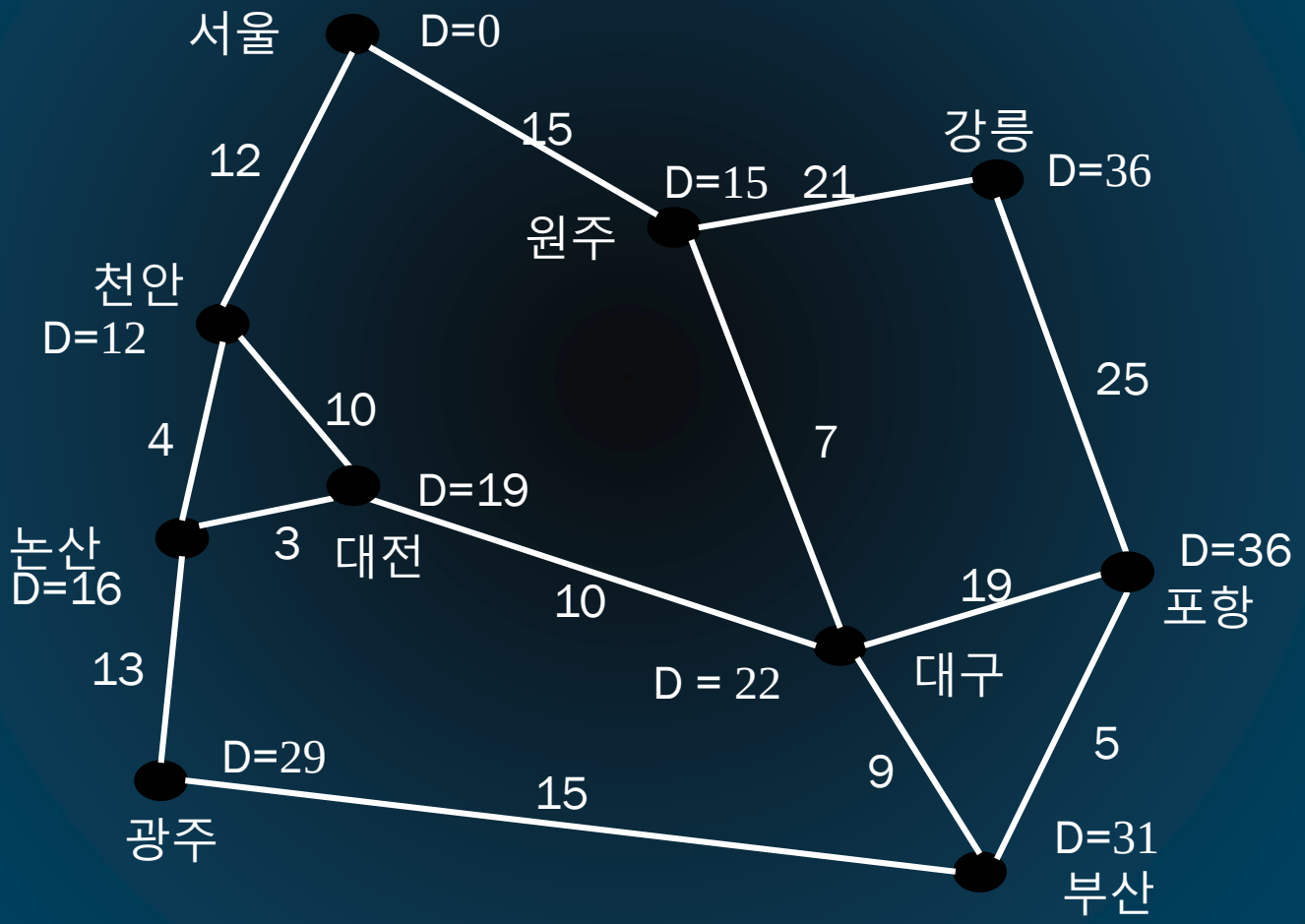
ShortestPath 알고리즘의 수행 과정(16)



ShortestPath 알고리즘의 수행 과정(17)



ShortestPath 알고리즘의 수행 과정(18)



시간복잡도

- while-루프가 (n-1)번 반복되고, 1회 반복될 때
  line 3에서 최소의 D[v]를 가진 점 v_{min}을 찾는데 O(n) 시간이 걸린다.
  - 왜냐하면 배열 D에서 최소값을 찾는 것이기 때문이다.
    또한 line 4에서도 v_{min}에 연결된 점의 수가 최대 (n-1)개이므로,
    각 D[w]를 갱신하는데 걸리는 시간은 O(n)이다.

- 따라서 시간복잡도는 (n-1)x{O(n)+O(n)} = $O(n^2)$이다.

응 용

- 맵퀘스트 (Mapquest)와 구글 (Google) 웹사이트의 지도 서비스
- 자동차 네비게이션
- 네트워크와 통신 분야
- 모바일 네트워크
- 산업 공학, 경영 공학의 OR(Operation Research)문제
- 로봇 공학
- 교통 공학
- VLSI 디자인 분야 등

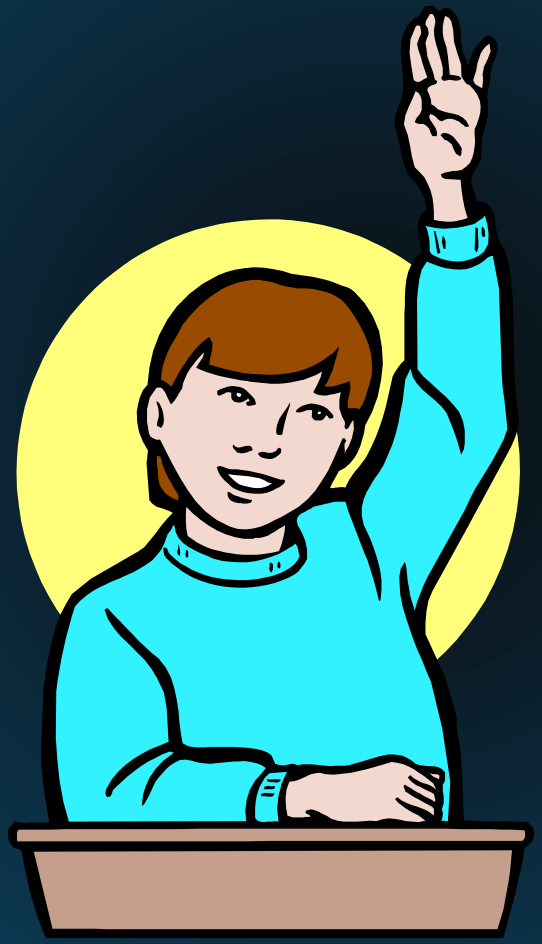
요 약

- 그리디 알고리즘은 (입력) 데이터 간의 관계를 고려하지 않고 수행 과정에서 '욕심내어' 최적값을 가진 데이터를 선택하며, 선택한 값들을 모아서 문제의 최적해를 찾는다.
- 그리디 알고리즘은 문제의 최적해 속에 부분 문제의 최적해가 포함되어 있고, 부분 문제의 해 속에 그 보다 작은 부분 문제의 해가 포함되어 있다.
  이를 최적 부분 구조 (Optimal Substructure) 또는 최적성 원칙 (Principle of Optimality)이라고 한다.
- 동전 거스름돈 문제를 해결하는 가장 간단한 방법은 남은 액수를 초과하지 않는 조건하에 가장 큰 액면의 동전을 취하는 것이다. 단, 일반적인 경우에는 최적해를 찾으나 항상 최적해를 찾지는 못한다.

요 약(2)

- 크러스컬 (Kruskal)의 최소 신장 트리 알고리즘은 가중치가 가장 작으면서 사이클을 만들지 않는 선분을 추가시키어 트리를 만든다. 시간복잡도는 O(mlogm)이다. 단, m은 그래프의 선분의 수이다.
- 프림 (Prim)의 최소 신장 트리 알고리즘은 최소의 가중치로 현재까지 만들어진 트리에 연결되는 선분을 트리에 추가시킨다. 시간복잡도는 $O(n^2)$이다.
- 다익스트라 (Dijkstra)의 최단 경로 알고리즘은 출발점으로부터 최단 거리가 확정되지 않은 점들 중에서 출발점으로부터 가장 가까운 점을 추가하고, 그 점의 최단 거리를 확정한다. 시간복잡도는 $O(n^2)$이다.

Q&A



BREAK TIME(2)

실습시간

1. 크러스컬 알고리즘을 파이선 구현하기
2. 프림 알고리즘을 파이선 구현하기
3. 크러스컬과 프림 알고리즘의 성능비교실험

- 숙제:
  1. 동전 거스름돈 문제를 파이선으로 구현하기
  2. 다익스트라 알고리즘을 파이선으로 구현하기

     → 서울-부산 도로정보를 고속도로 실제거리로 입력해서 최단 경로 구하기,