12주차 1차시 그리디 알고리즘의 이해 I

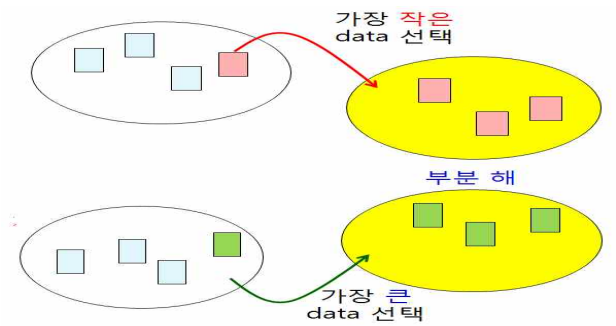
【학습목표】

1. 최소 신장 트리 알고리즘을 이해할 수 있다.
2. 최단 경로 찾기 알고리즘을 이해할 수 있다.

학습내용1 : 최소 신장 트리

1. 신장 트리 문제

- 신장 트리 문제는 그리디 알고리즘(최적화 문제)으로 해결 한다.
- 최적화 문제 : 가능한 해들 중에서 가장 좋은 (최대 또는 최소) 해를 찾는 문제
- 욕심쟁이 방법, 탐욕적 방법, 탐욕 알고리즘 등으로 불리기도 한다.
- 그리디 알고리즘은 (입력) 데이터 간의 관계를 고려하지 않고 수행 과정에서 '욕심내어' 최소값 또는 최대값을 가진 데이터를 선택한다.
- 이러한 선택을 '근시안적'인 선택이라고 말하기도 한다.

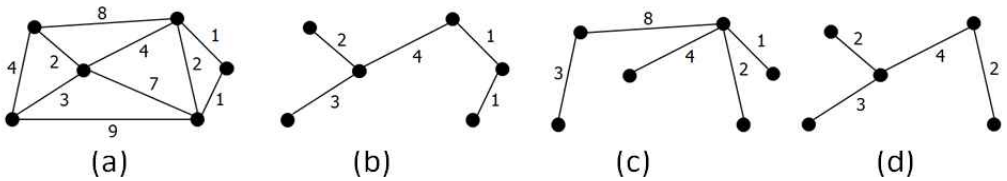


그리디 알고리즘 수행 과정

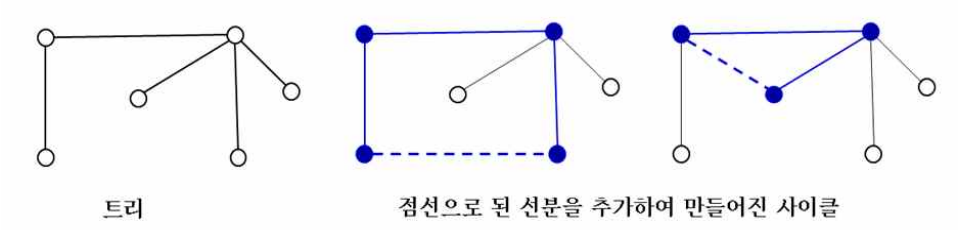
- 그리디 알고리즘은 일단 한번 선택하면, 이를 절대로 번복하지 않는다.
- 즉, 선택한 데이터를 버리고 다른 것을 취하지 않는다.
- 이러한 특성 때문에 대부분의 그리디 알고리즘들은 매우 단순하며, 또한 제한적인 문제들만이 그리디 알고리즘으로 해결된다.

2. 최소 신장 트리 (Minimum Spanning Tree)

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



- 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.



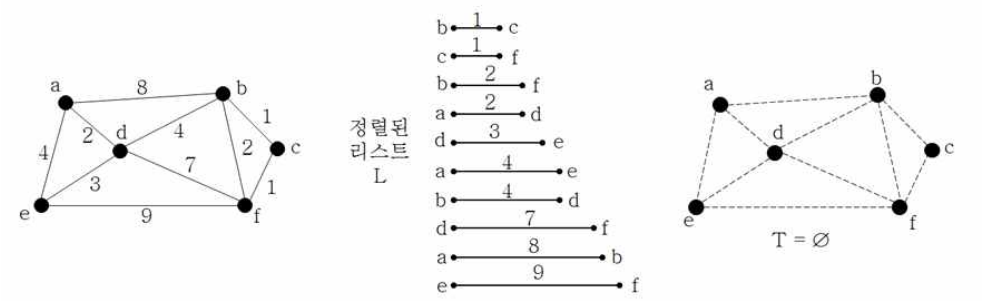
- 최소 신장 트리를 찾는 대표적인 그리디 알고리즘으로는 크러스컬 (Kruskal)과 프림 (Prim) 알고리즘이 있다.
알고리즘의 입력은 1개의 연결요소 (connected component)로 된 가중치 그래프이다.
- 크러스컬 알고리즘은 가중치가 가장 작은 선분이 사이클을 만들지 않을 때에만 '욕심내어' 그 선분을 추가시킨다. 다음은 크러스컬의 최소 신장 트리 알고리즘이다.

* 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=∅ // 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.

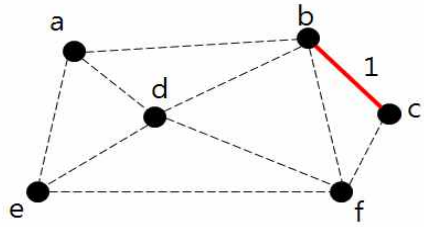
- Line 1: 모든 선분들을 가중치의 오름차순으로 정렬한다. 정렬된 선분들의 리스트를 L이라고 하자.
- Line 2: T를 초기화시킨다. 즉, T에는 아무 선분도 없는 상태에서 시작된다.
- Line 3~8의 while-루프는 T의 선분 수가 (n-1)이 될 때까지 수행되는데 1번 수행될 때마다 L에서 가중치가 가장 작은 선분 e를 가져온다. 단, 가져온 선분 e는 L에서 삭제되어 다시는 고려되지 않는다.
- Line 5~8: 가져온 선분 e를 T에 추가되어 사이클을 만들지 않으면 e를 T에 추가시키고, 사이클을 만들면 선분 e를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

다음의 그래프에서 KruskalMST 알고리즘이 최소 신장 트리를 찾는 과정을 살펴보자.



리스트 L

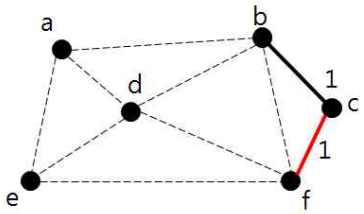
b 1 c
c 1 f
b 2 f
a 2 d
d 3 e
a 4 e
b 4 d
d 7 f
a 8 b
e 9 f



선분 (b,c) 추가

리스트 L

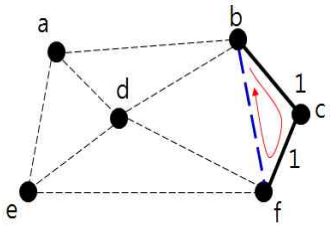
c 1 f
b 2 f
a 2 d
d 3 e
a 4 e
b 4 d
d 7 f
a 8 b
e 9 f



선분 (c,f) 추가

리스트 L

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

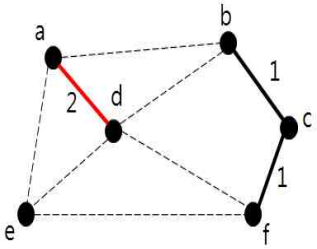


사이클 b-c-f-b

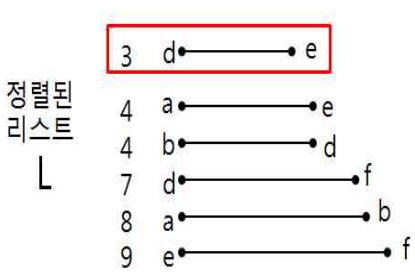
선분 (b,f) 버림

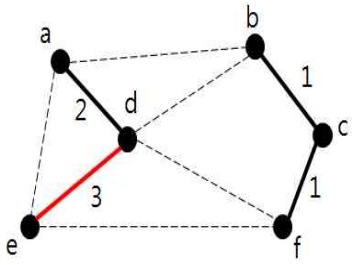
정렬된 리스트 L

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

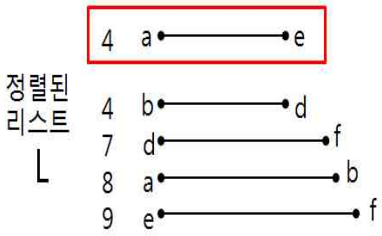


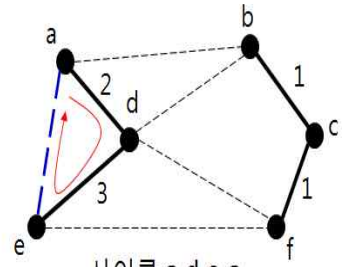
선분 (a,d) 추가





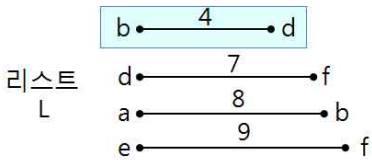
선분 (d,e) 추가

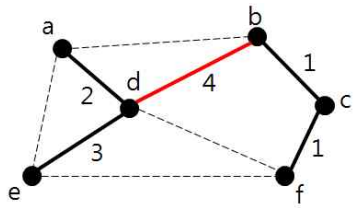




사이클 a-d-e-a

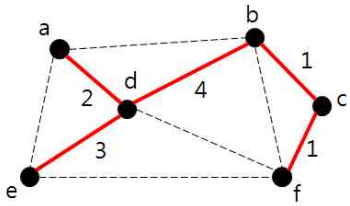
선분 (a,e) 버림





선분 (b,d) 추가

최종해



3. Kruskal 알고리즘의 시간복잡도

- Line 1: 정렬하는데 O(mlogm) 시간. 단, m은 입력 그래프에 있는 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(mlogm)+O(mlog*m) = O(mlogm)이다.

학습내용2 : 최단 경로 찾기

1. 최단 경로 (Shortest Path) 문제

- 최단 경로 (Shortest Path) 문제는 주어진 가중치 그래프에서 어느 한 출발점에서 또 다른 도착점까지의 최단 경로를 찾는 문제이다.
- 최단 경로를 찾는 가장 대표적인 알고리즘은 다이스트라 (Dijkstra) 최단 경로 알고리즘이며, 이 또한 그리디 알고리즘이다.
  ☞ 단일 출발점 최단 경로.(다이스트라 알고리즘)
  ☞ 모든 쌍 최단 경로.(플로이드 알고리즘)

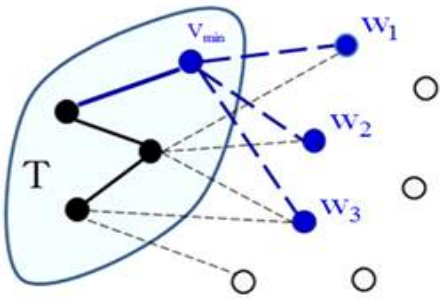
- 프림의 최소 신장트리 알고리즘과 거의 흡사한 과정으로 진행 된다. 2가지 차이점:
 1. 프림 알고리즘은 임의의 점에서 시작하나, 다이스트라 알고리즘은 주어진 출발점에서 시작한다.
 2. 프림 알고리즘은 트리에 하나의점(선분)을 추가시킬 때 현재 상태의 트리에서 가장 가까운 점을 추가시킨다. 그러나 다이스트라 알고리즘은 출발점 으로 부터 최단 거리가 확정되지 않은 점들 중에서 출발점으로부터 가장 가까운 점을 추가하고, 그 점의 최단 거리를 확정한다.

* 다이스트라 알고리즘(단일 출발점 최단 경로)

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

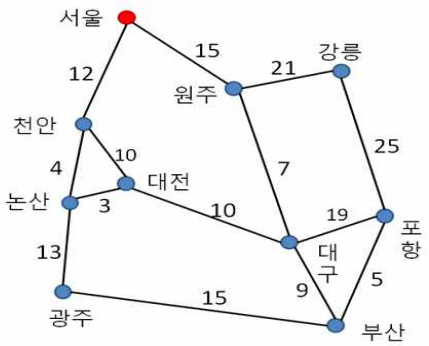
- 알고리즘에서 배열 D[v]는 출발점 s로부터 점 v까지의 거리를 저장하는데 사용하고, 최종적으로는 출발점 s로부터 점 v까지의 최단 거리를 저장하게 된다. Line 1에서는 출발점 s의 D[s]=0으로, 또 다른 각 점 v에 대해서 D[v]=∞로 초기화시킨다.
- Line 2~4의 while-루프는 (n-1)회 수행된다. 현재까지 s로부터 최단 거리가 확정된 점들의 집합을 T라고 놓으면, V-T는 현재까지 s로부터 최단 거리가 확정되지 않은 점들의 집합이다. 따라서 V-T에 속한 각 점 v에 대해서 D[v]가 최소인 점 v_{min}을 선택하고, v_{min}의 최단 거리를 확정시킨다. 즉, D[v_{min}] ≤ D[v], v∈V-T이다. '확정한다는 것'은 2가지의 의미를 갖는다.
- D[v_{min}]이 확정된 후에는 다시 변하지 않는다.
- 점 v_{min}이 T에 포함된다

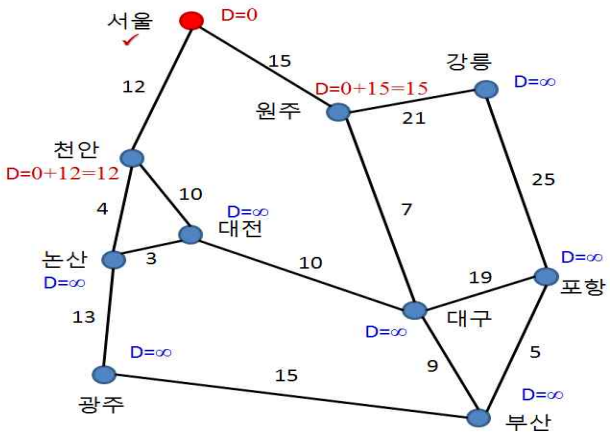
- Line 4: V-T에 속한 점들 중 v_{min}을 거쳐 감 (경유함)으로서 s로부터의 거리가 현재보다 더 짧아지는 점 w가 있으면, 그 점의 D[w]를 갱신한다.
- 다음 그림은 v_{min}이 T에 포함된 상태를 보이고 있는데, v_{min}에 인접한 점 w_1, w_2, w_3 각각에 대해서 만일 (D[v_{min}]+선분(v,w_i)의 가중치)<D[w_i]이면, D[w_i] = (D[v_{min}]+선분(v_{min},w_i)의 가중치)로 갱신한다.

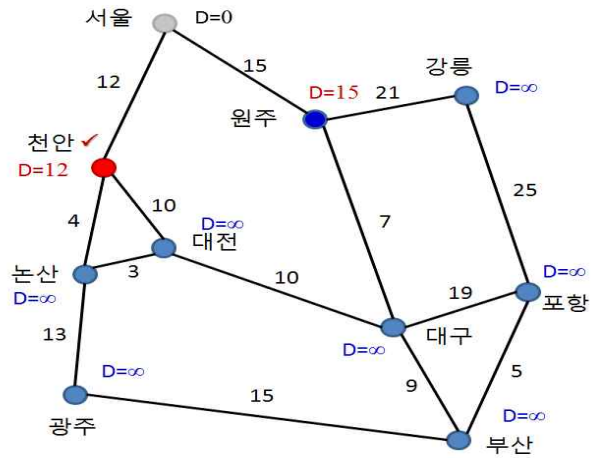


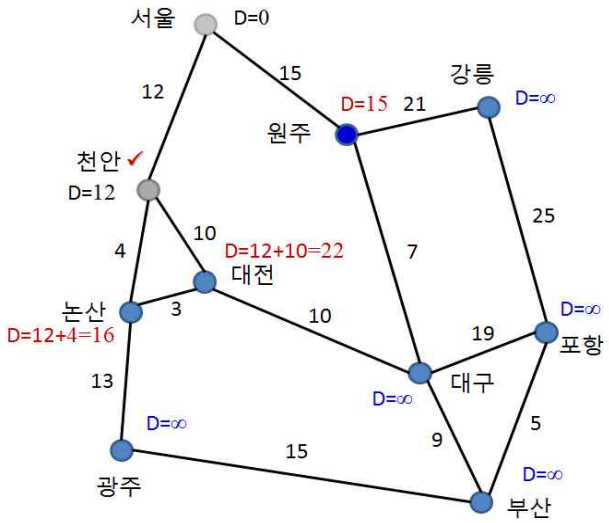
- 마지막으로 line 5에서 배열 D를 리턴 한다.

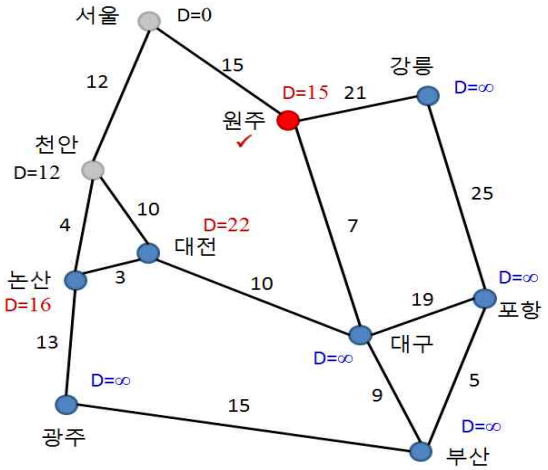
- ShortestPath 알고리즘의 수행 과정
- 단, 출발점은 서울이다.

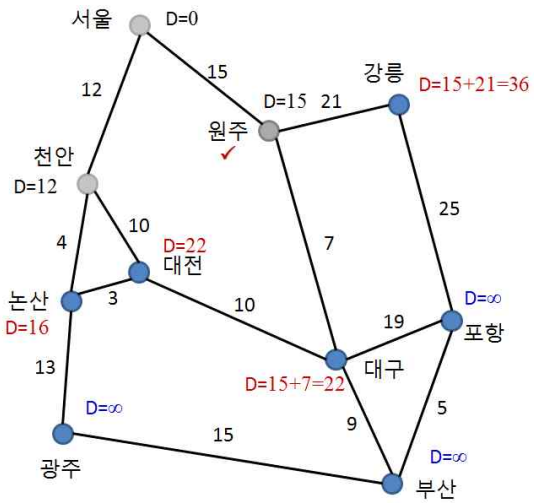


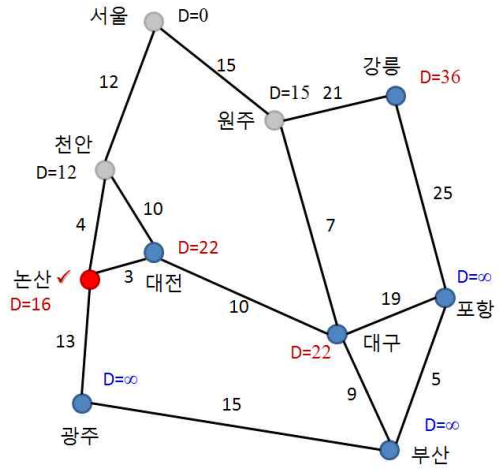


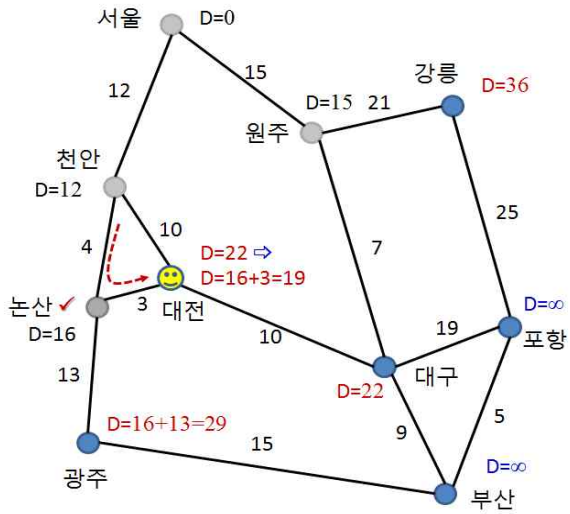

서울
D=0
15
12
강릉
D=15
21
D=∞
원주
천안
D=12
25
10
4
D=∞
대전
7
논산
3
10
D=∞
19
포항
D=∞
13
대구
D=∞
D=∞
5
9
15
광주
D=∞
부산

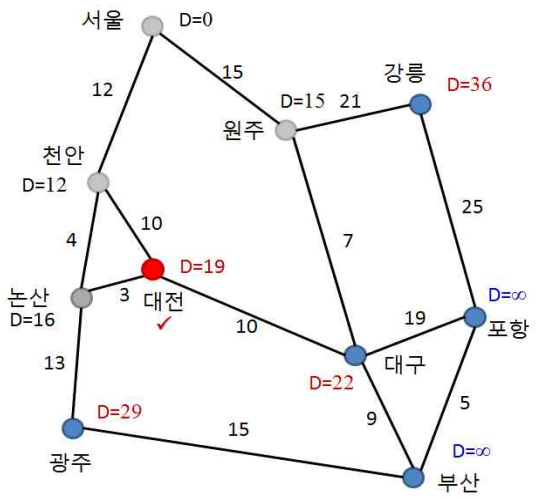

서울
D=0
15
12
강릉
D=15
21
D=∞
원주
천안
D=12
25
10
4
D=12+10=22
7
대전
논산
3
10
D=∞
D=12+4=16
19
포항
13
대구
D=∞
D=∞
5
9
15
광주
D=∞
부산

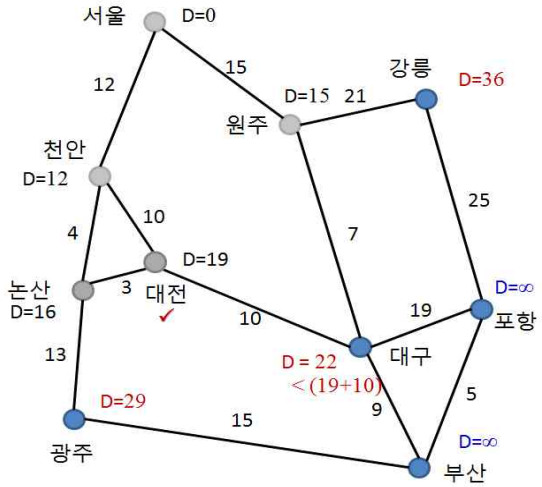

서울
D=0
15
12
강릉
D=15
21
D=∞
원주
천안
D=12
25
10
4
D=22
7
대전
논산
3
10
D=∞
D=16
19
포항
13
대구
D=∞
D=∞
5
9
15
광주
D=∞
부산

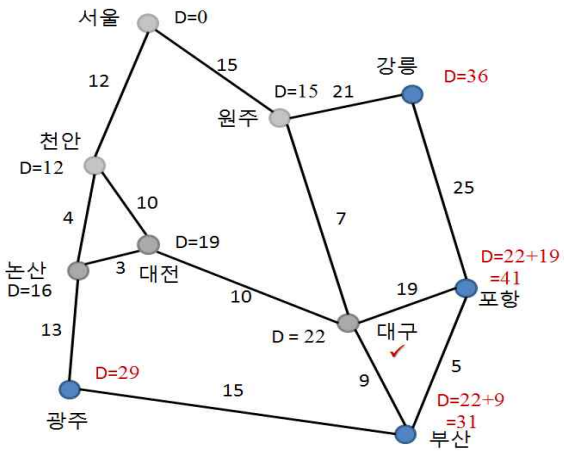

서울
D=0
12
15
강릉
D=15+21=36
D=15
21
원주
천안
D=12
4
10
D=22
대전
7
25
논산
3
10
D=∞
D=16
19
포항
13
대구
D=15+7=22
D=∞
9
5
15
광주
D=∞
부산

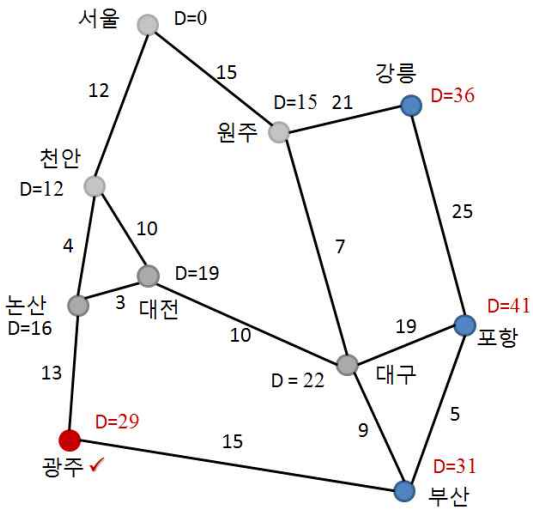

서울
D=0
12
15
강릉
D=36
D=15
21
원주
천안
D=12
25
4
10
7
D=22
논산
3
대전
D=∞
D=16
19
포항
10
13
대구
D=22
D=∞
9
5
15
광주
D=∞
부산

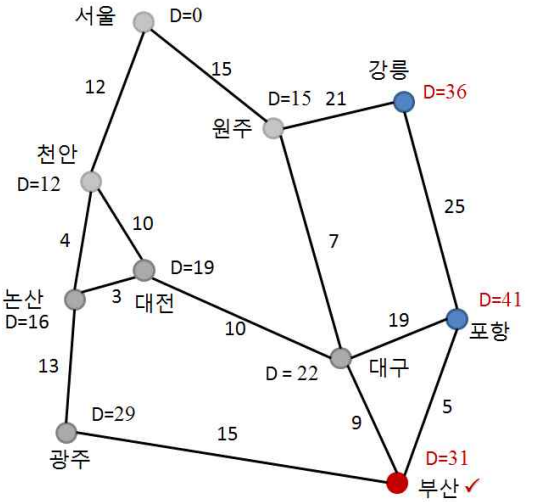

서울
D=0
15
12
강릉
D=36
D=15
21
원주
천안
D=12
25
10
4
7
D=22 ⇨
D=16+3=19
논산
3
대전
D=∞
D=16
19
포항
10
13
대구
D=22
D=16+13=29
9
5
15
D=∞
광주
부산

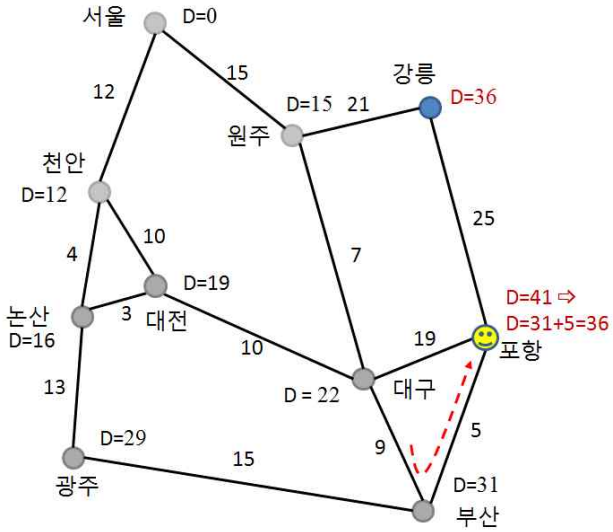

서울
D=0
15
12
강릉
D=36
D=15
21
원주
천안
D=12
25
10
4
7
D=19
논산
D=16
3
대전
D=∞
10
19
포항
13
대구
D=22
5
D=29
9
15
D=∞
광주
부산

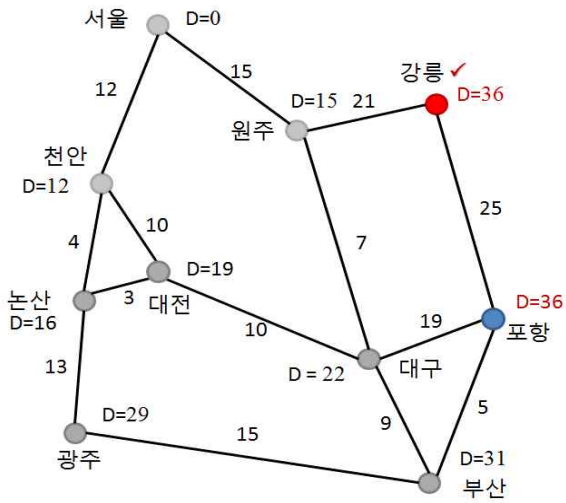

서울
D=0
15
12
강릉
D=36
D=15
21
원주
천안
D=12
25
10
4
7
D=19
논산
D=16
3
대전
D=∞
10
19
포항
13
D = 22
< (19+10)
대구
5
D=29
9
15
D=∞
광주
부산

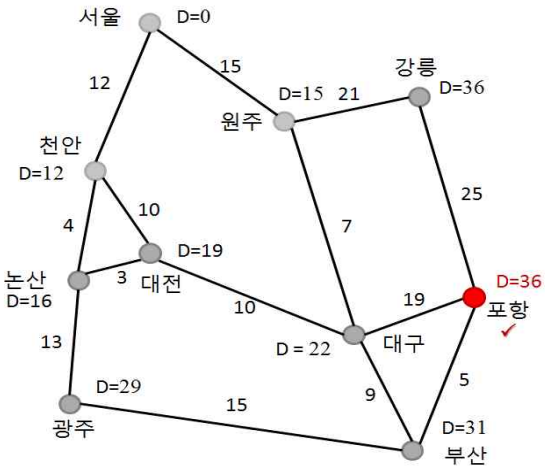

서울
D=0
15
12
강릉
D=36
D=15
21
원주
천안
D=12
25
10
4
7
D=19
논산
D=16
3
대전
D=22+19
=41
10
19
포항
13
D = 22
대구
5
D=29
9
15
D=22+9
=31
광주
부산

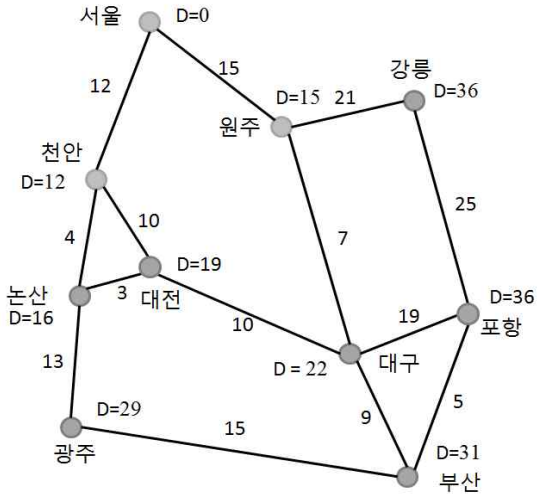

서울
D=0
15
12
강릉
D=15
21
D=36
원주
천안
D=12
25
10
4
7
D=19
논산
D=16
3
대전
D=41
19
10
포항
13
D = 22
대구
5
D=29
9
15
D=31
광주 ✓
부산


서울
D=0
15
12
강릉
D=15
21
D=36
원주
천안
D=12
25
10
4
7
D=19
논산
D=16
3
대전
D=41
19
10
포항
13
D = 22
대구
5
D=29
9
15
D=31
광주
부산 ✓


서울
D=0
15
12
강릉
D=15
21
D=36
원주
천안
D=12
25
10
4
7
D=19
논산
D=16
3
대전
D=41 ⇨
D=31+5=36
19
10
포항
13
D = 22
대구
5
D=29
9
15
D=31
광주
부산


서울
D=0
15
12
강릉 ✓
D=15
21
D=36
원주
천안
D=12
25
10
4
7
D=19
논산
D=16
3
대전
D=36
10
19
포항
13
D = 22
대구
5
D=29
9
15
광주
D=31
부산


서울
D=0
15
12
강릉
D=15
21
D=36
원주
천안
D=12
25
10
4
7
D=19
논산
D=16
3
대전
D=36
10
19
포항 ✓
13
D = 22
대구
5
D=29
9
15
광주
D=31
부산


서울
D=0
15
12
강릉
D=15
21
D=36
원주
천안
D=12
25
10
4
7
D=19
논산
D=16
3
대전
D=36
10
19
포항
13
D = 22
대구
5
D=29
9
15
광주
D=31
부산

2. 시간 복잡도

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

【학습정리】

1. 최소 신장 트리
- 신장 트리 문제는 그리디 알고리즘(최적화 문제)으로 해결 한다.
- 최적화 문제: 가능한 해들 중에서 가장 좋은 (최대 또는 최소) 해를 찾는 문제
- 욕심쟁이 방법, 탐욕적 방법, 탐욕 알고리즘 등으로 불리기도 한다.
- 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 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.

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

2. 최단 경로 찾기
- 최단 경로 (Shortest Path) 문제는 주어진 가중치 그래프에서 어느 한 출발점에서 또 다른 도착점까지의 최단 경로를 찾는 문제이다.
- 최단 경로를 찾는 가장 대표적인 알고리즘은 다이스트라 (Dijkstra) 최단 경로 알고리즘이며, 이 또한 그리디 알고리즘이다
  ☞ 단일 출발점 최단 경로.(다이스트라 알고리즘)
  ☞ 모든 쌍 최단 경로.(플로이드 알고리즘)