7. 그래프 알고리즘 2

한국외국어대학교
고 석 훈

목차

7.5 최단 경로 찾기

- 최단 경로(Shortest Path) 문제
  - 주어진 가중치 그래프에서 어느 한 출발점에서 또 다른 도착점까지의 최단 경로를 찾는 문제이다.

- 최단 경로를 찾는 가장 대표적인 알고리즘은 다익스트라(Dijkstra) 알고리즘이며, 이는 그리디 알고리즘이다.

- Dijkstra의 최단 경로 알고리즘은 Prim의 최소 신장 트리 알고리즘과 유사한 과정으로 진행되지만, 2가지 차이점이 있다.
  - Prim 알고리즘은 임의의 점에서 시작하나,
    Dijkstra 알고리즘은 주어진 출발점에서 시작한다.
  - Prim 알고리즘은 트리에 간선을 추가시킬 때,
    현재 상태의 트리에서 가장 가까운 정점을 추가시킨다.

    그러나, Dijkstra 알고리즘은 출발점으로부터 최단 거리가 확정되지 않은 점들 중에서 출발점으로부터 가장 가까운 점을 찾아 추가하고 그 점의 최단 거리를 확정한다.

Dijkstra 알고리즘

ShortestPath(G, s)

입력: 가중치 그래프 $G = (V, E), |V| = n, |E| = m$

출력: 출발점 s로부터 n개의 점까지 각각 최단 거리를 저장한 배열 D,

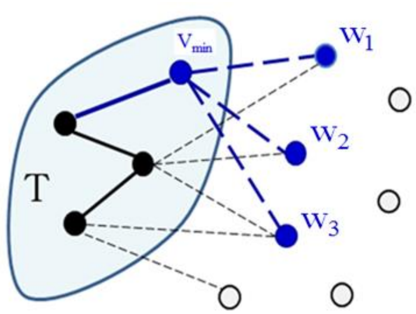
(배열 $D[v]$는 출발점 s로부터 점 v까지의 거리를 의미)

1. 배열 D를 ∞로 초기화시킨다. 단 $v_{min} = s, D[s] = 0$으로 확정한다.
2. $while$ (최단 거리가 확정되지 않은 정점이 있으면) {
3. 정점 v_{min}과 인접한 각 정점 w에 대해서 $D[w]$를 갱신한다.
   (기존 $D[w]$와 $D[v_{min}] +$ 선분 (v, w)의 가중치 중에 작은 값 선택)
4. 확정된 정점과 인접한 모든 확정되지 않은 정점 v의 집합에서
   최소의 $D[v]$의 값을 가진 정점 v_{min}을 선택하고,
   출발점 s로부터 점 v_{min}까지의 최단 거리 $D[v_{min}]$을 확정한다.
5. $return\ D$

ShortestPath 알고리즘 설명

- Line 1:
  - 배열 $D[v]$는 출발점 s로부터 점 v까지의 거리의 중간 결과를 저장하는데 사용하다가, 최종적으로는 출발점 s로부터 점 v까지의 최단 거리를 저장한다.
  - 출발점 s를 v_{min}으로 정하고, 출발점 s의 $D[s] = 0$으로 확정한다. 또 다른 각 점 v에 대해서 $D[v] = \infty$로 초기화한다.
  - v_{min}의 최단 거리를 '확정'하는 것은 다음 2가지의 의미를 갖는다.
    - $D[v_{min}]$이 확정된 후에는 다시 변하지 않는다.
    - 점 v_{min}을 최단 거리가 확정된 정점 집합 T에 포함시킨다.

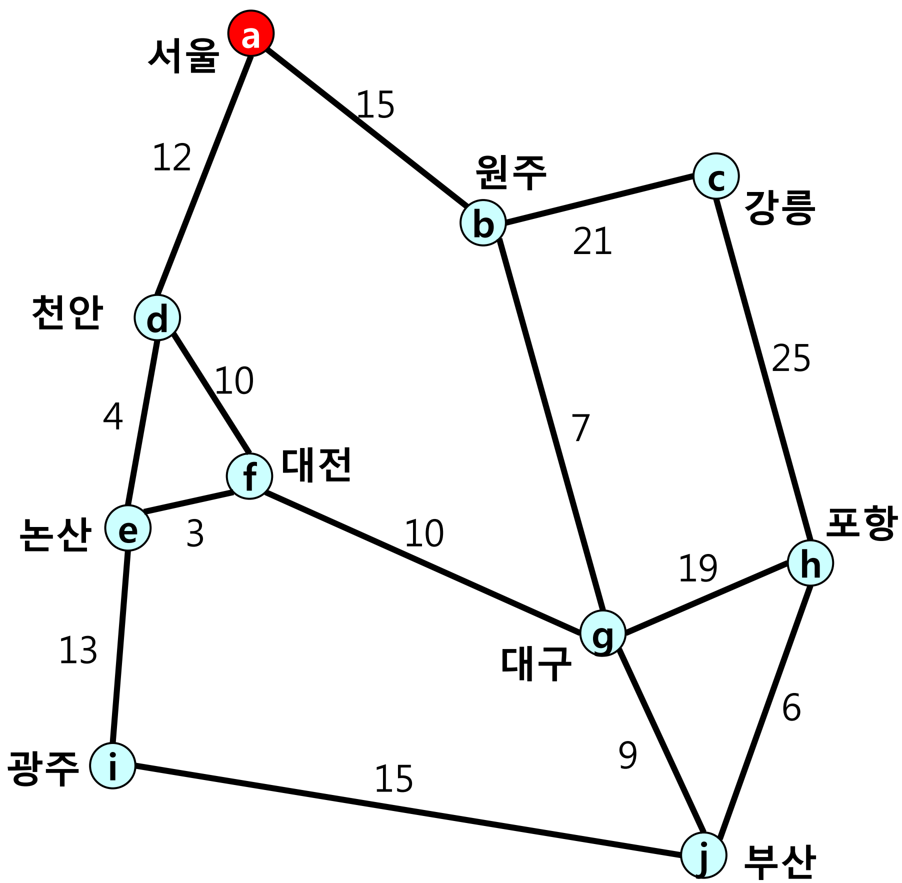
- Line 2~3:
  - $while$ 루프는 $(n-1)$회 수행된다.
  - 현재까지 s로부터 최단 거리가 확정된 점들의 집합을 T라 하면, $V-T$는 현재까지 최단 거리가 확정되지 않은 점들의 집합이다.
  - $V-T$에 속한 점들 중 v_{min}과 인접한 점 w의 $D[w]$를 갱신한다.
  - 다음 그림은 v_{min}이 T에 포함된 상태를 보이고 있는데,
    v_{min}에 인접한 점 w_1, w_2, w_3 각각에 대해서
    만일 $(D[v_{min}] +$ 선분(v_{min}, w_i)의 가중치$) < D[w_i]$이면,
    $D[w_i] = (D[v_{min}] +$ 선분(v_{min}, w_i)의 가중치$)$로 갱신한다.

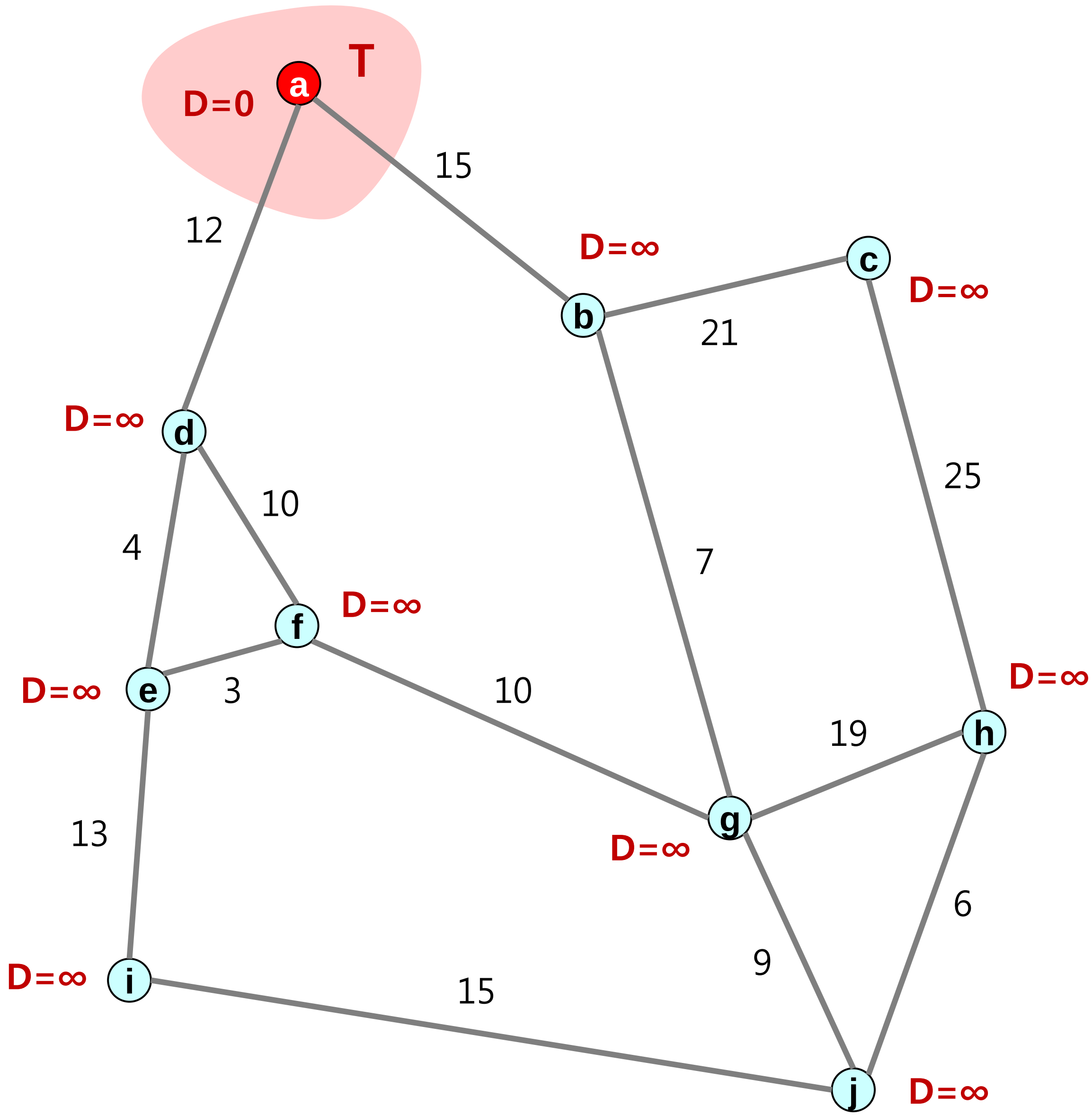


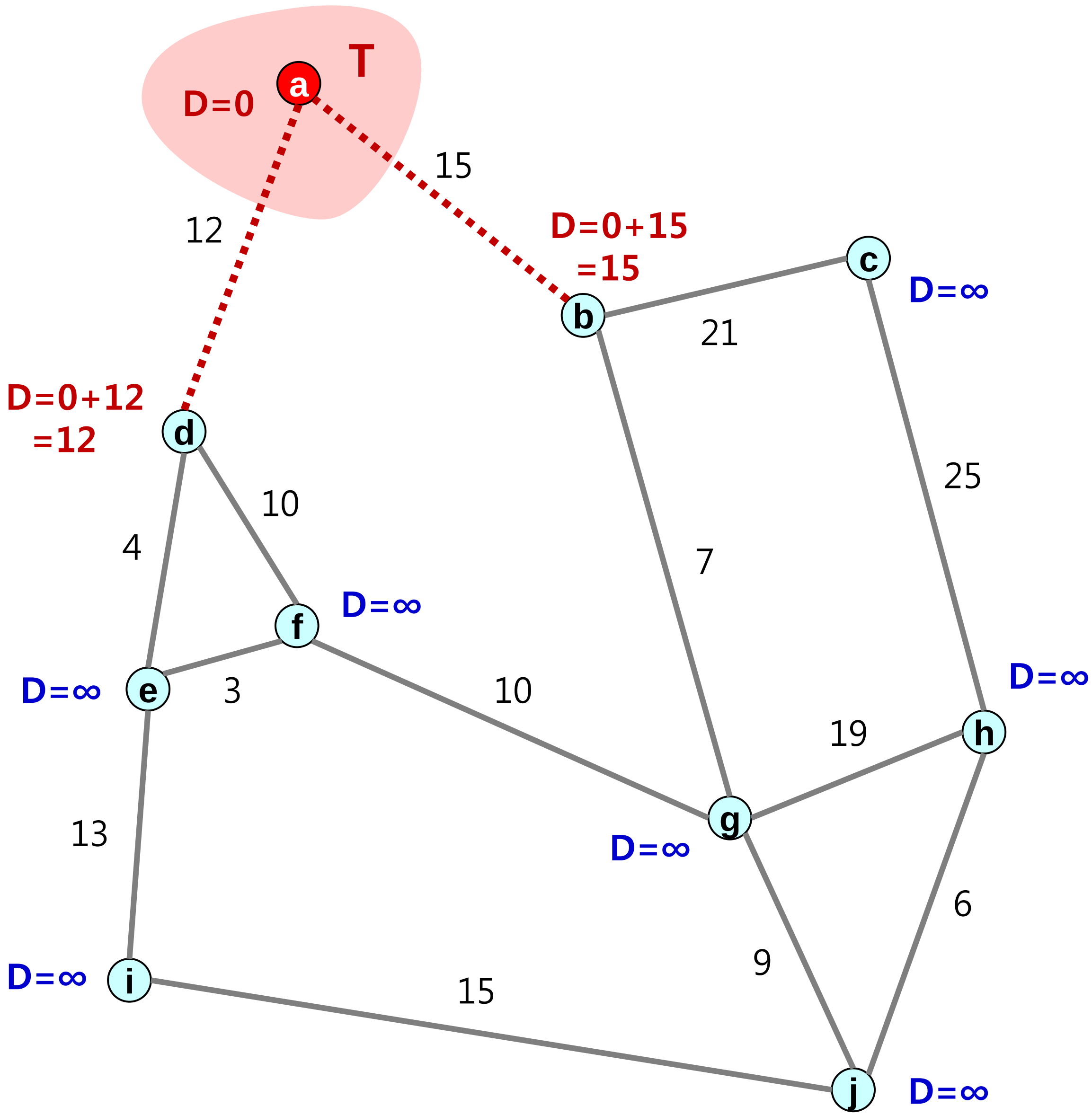
- Line 4:
  - $V-T$에 속한 각 점 v에 대해서 $D[v]$가 최소인 점 v_{min}을 선택하고, v_{min}의 최단 거리를 확정시킨다. 즉, $D[v_{min}] \leq D[v], v \in V-T$이다.
    - 이때, v_{min}의 후보는 집합 T에 속한 확정된 정점 t에 인접한 정점v로 한정할 수 있다.
  - v_{min}의 최단 거리를 '확정한다는 것'은 다음 2가지 의미를 갖는다.
    - $D[v_{min}]$이 확정된 후에는 다시 변하지 않는다.
    - 점 v_{min}을 최단 거리가 확정된 정점 집합 T에 포함시킨다.
- Line 5:
  - 배열 D를 리턴 한다.

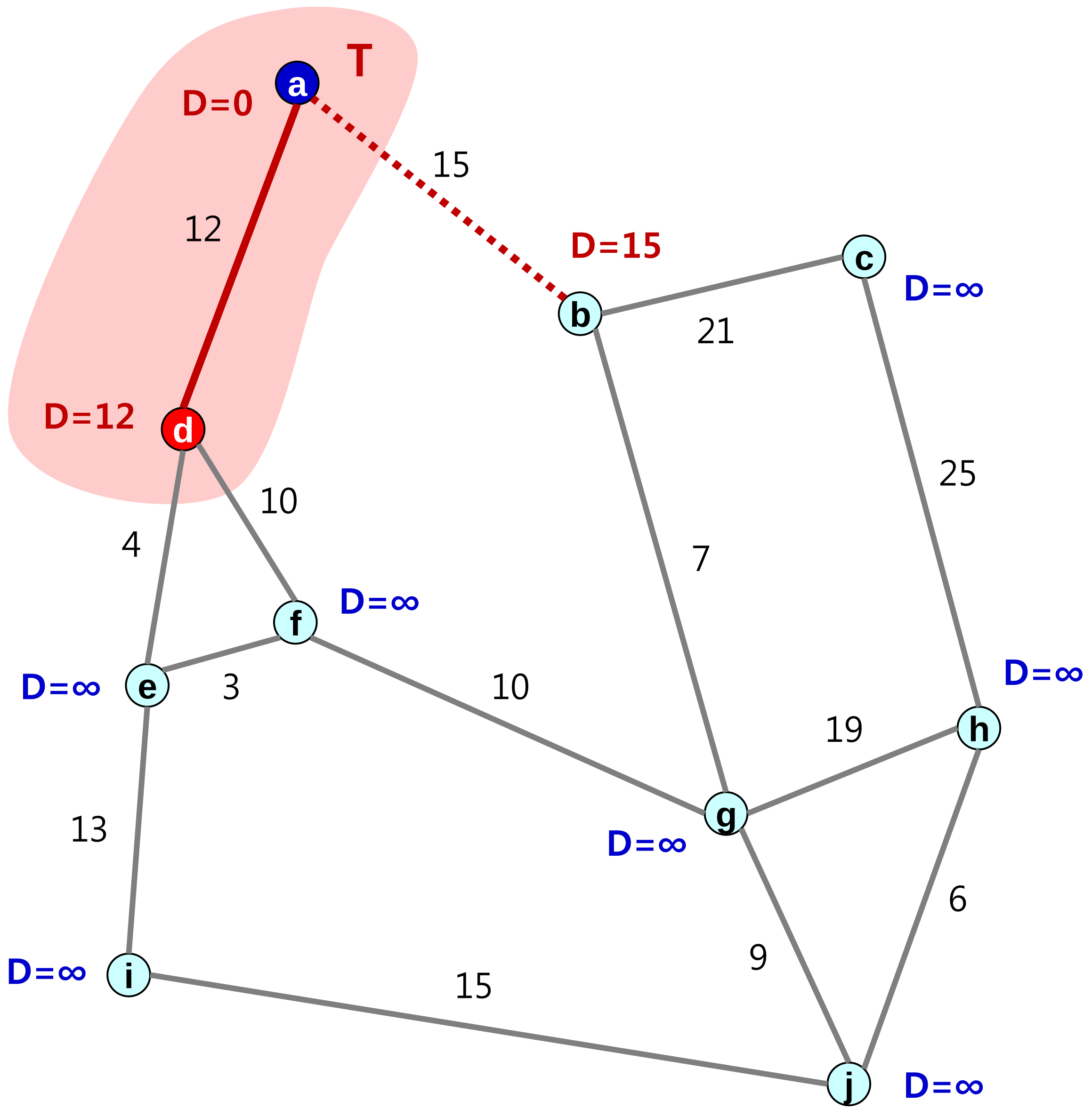
ShortestPath 알고리즘의 수행 과정

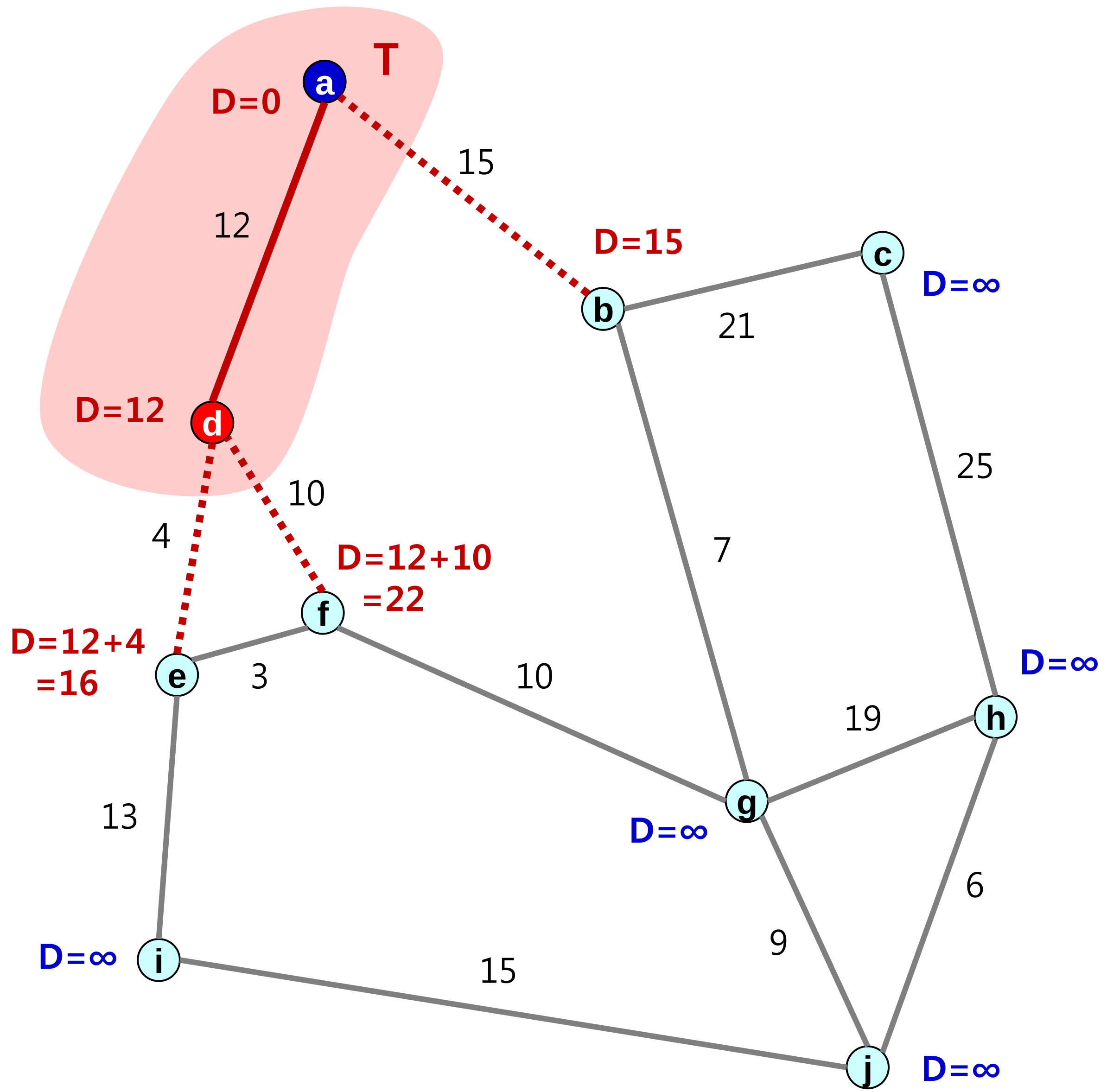
- 출발점은 서울

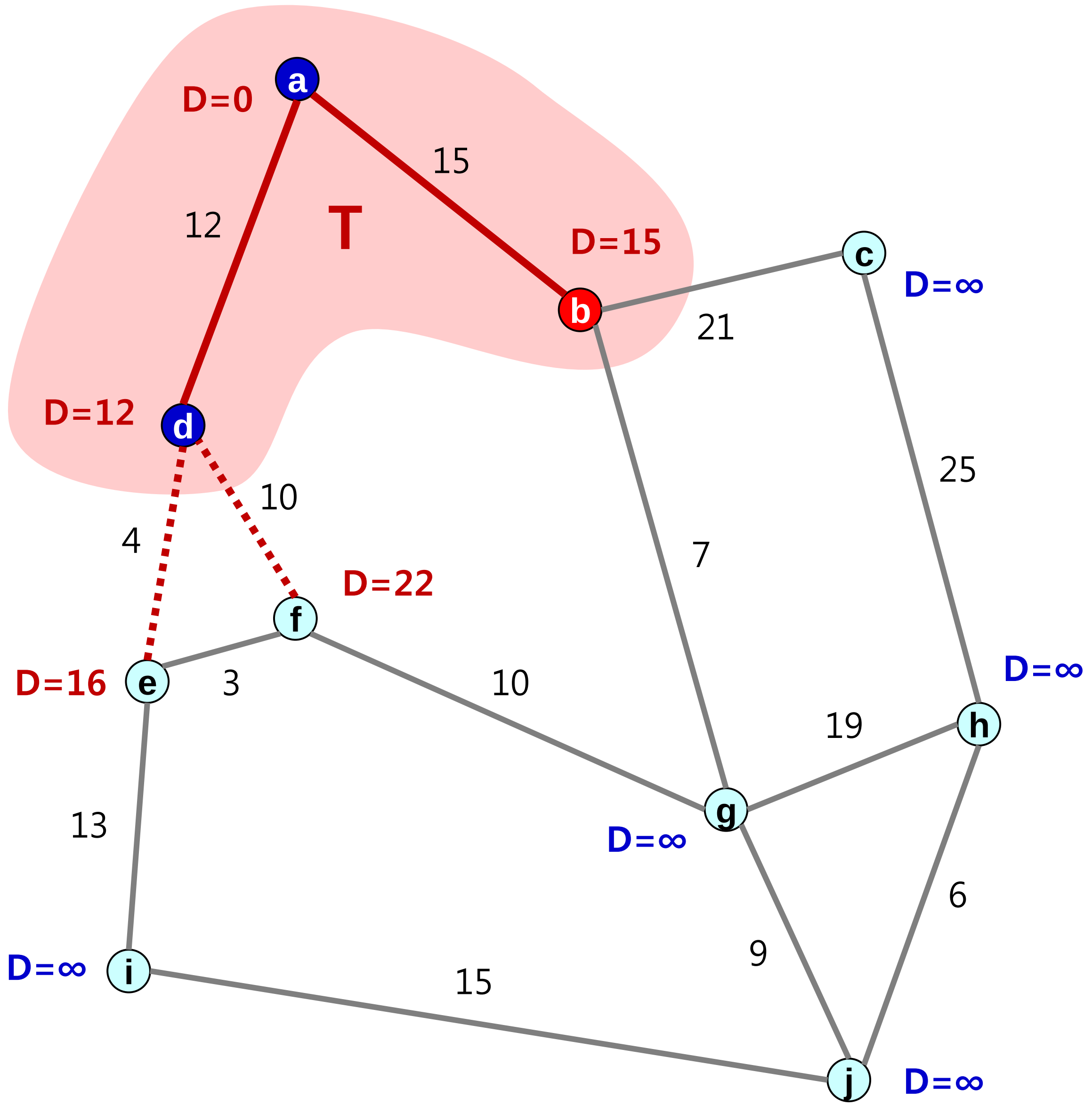


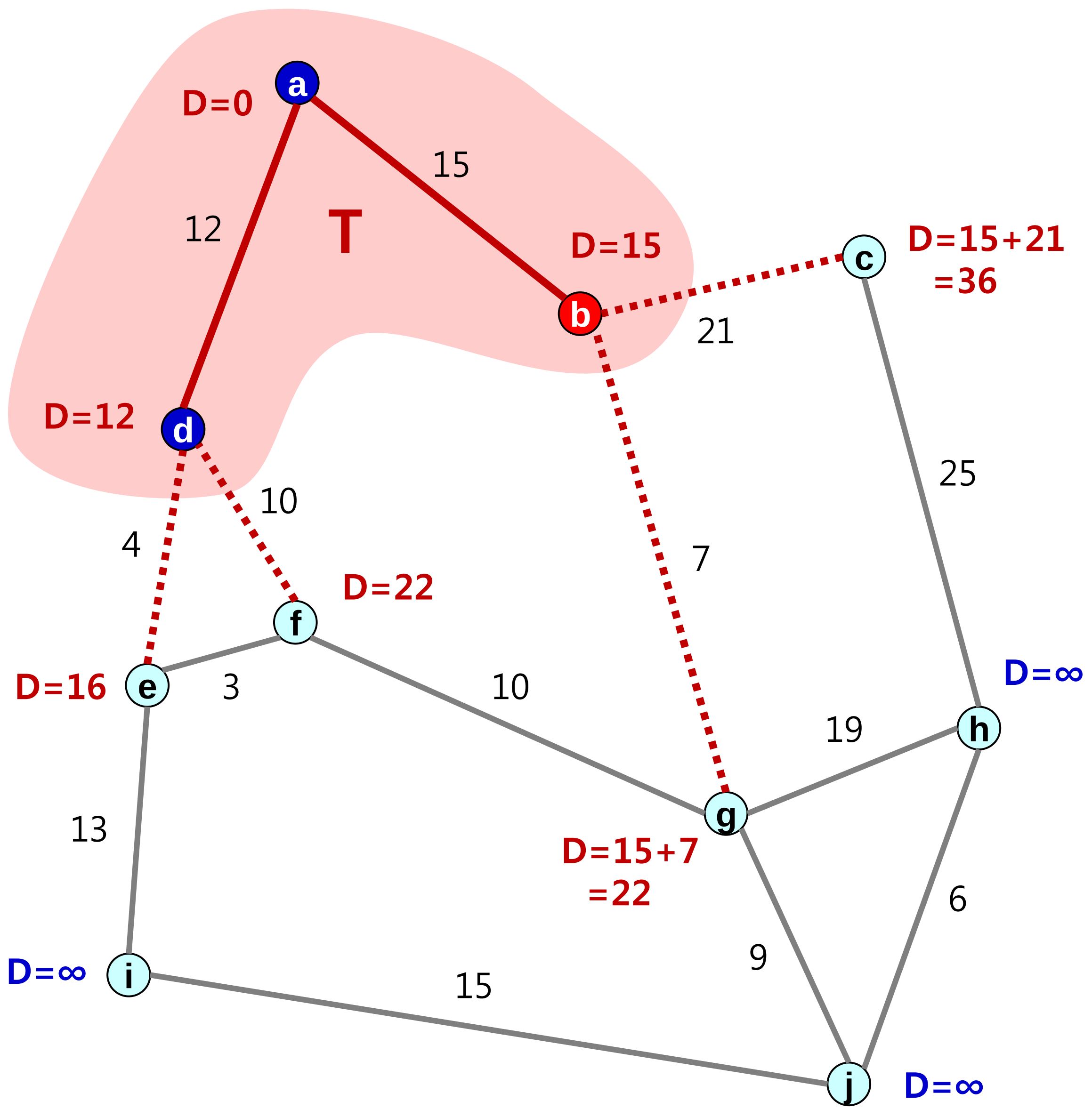

T
a
D=0
15
12
b
D=∞
c
D=∞
21
d
D=∞
10
25
4
7
f
D=∞
e
D=∞
3
10
D=∞
h
19
13
g
D=∞
6
9
i
D=∞
15
j
D=∞

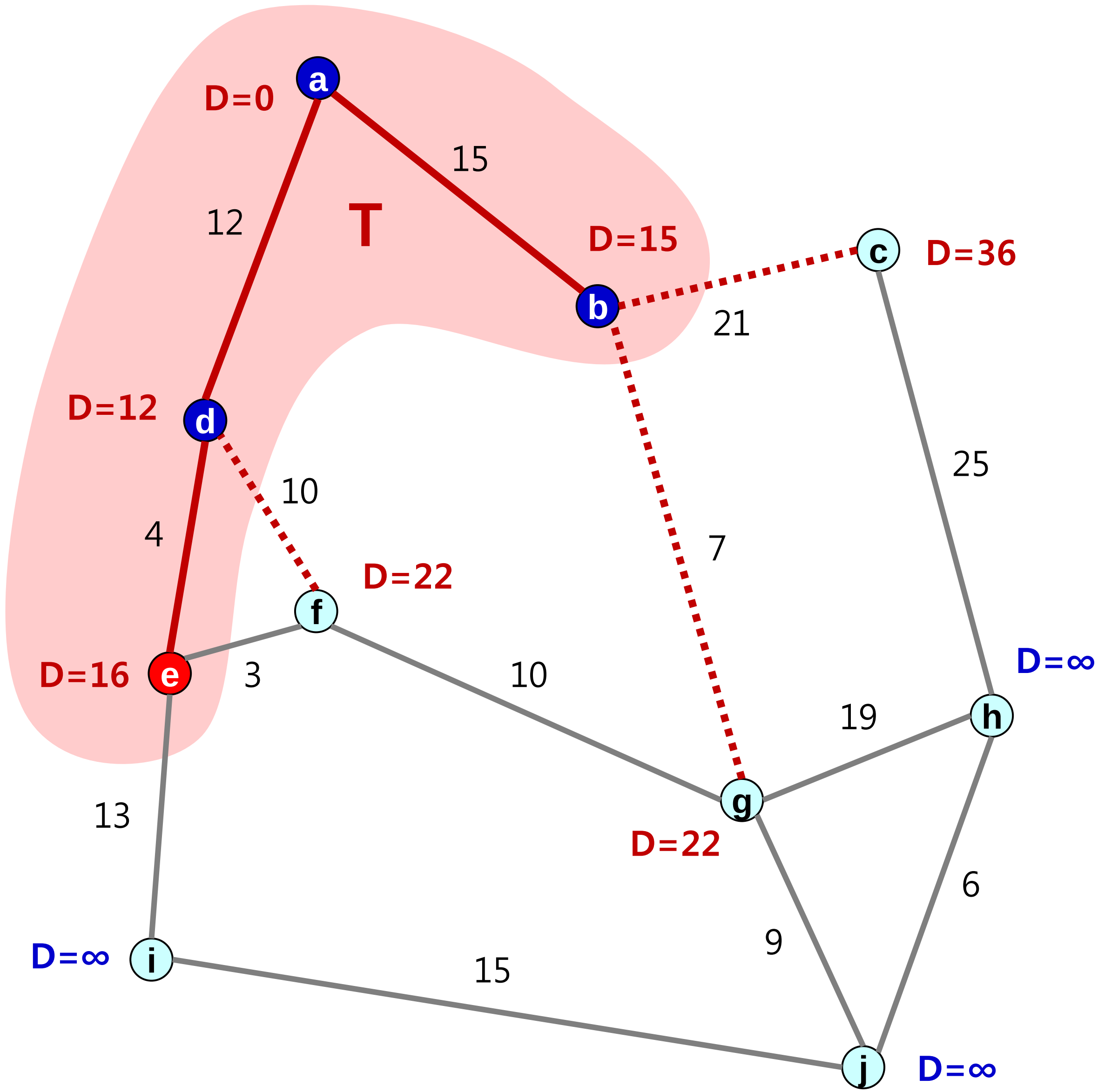

T
a
D=0
15
12
D=0+15
=15
c
D=∞
b
21
D=0+12
=12
d
25
10
4
7
f
D=∞
D=∞
D=∞
e
3
10
19
h
g
13
D=∞
6
9
D=∞
i
15
j
D=∞

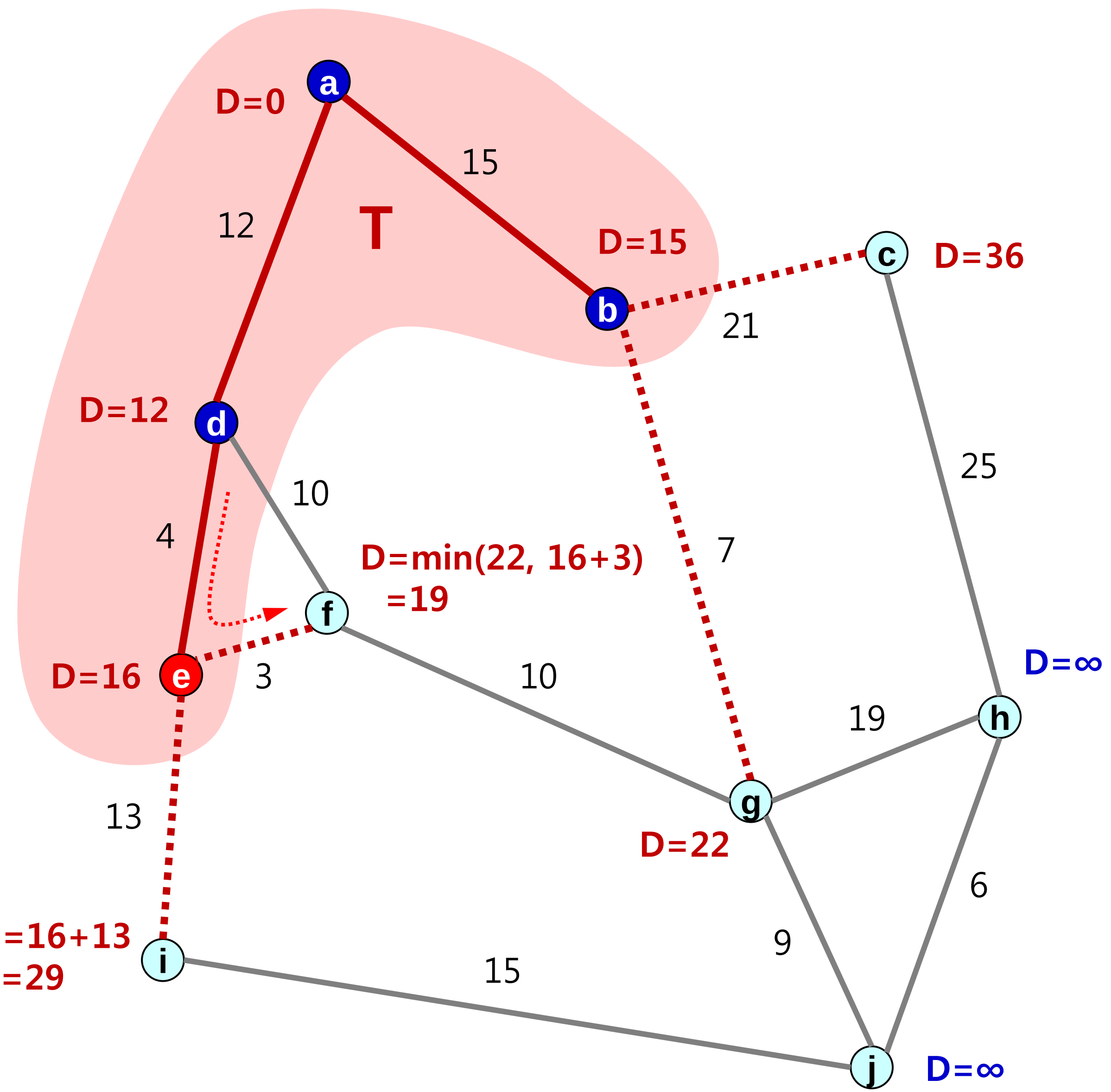

T
a
D=0
15
12
D=15
b
c
D=∞
21
D=12
d
25
10
4
7
f
D=∞
D=∞
D=∞
e
3
10
19
h
13
g
D=∞
6
9
D=∞
i
15
j
D=∞

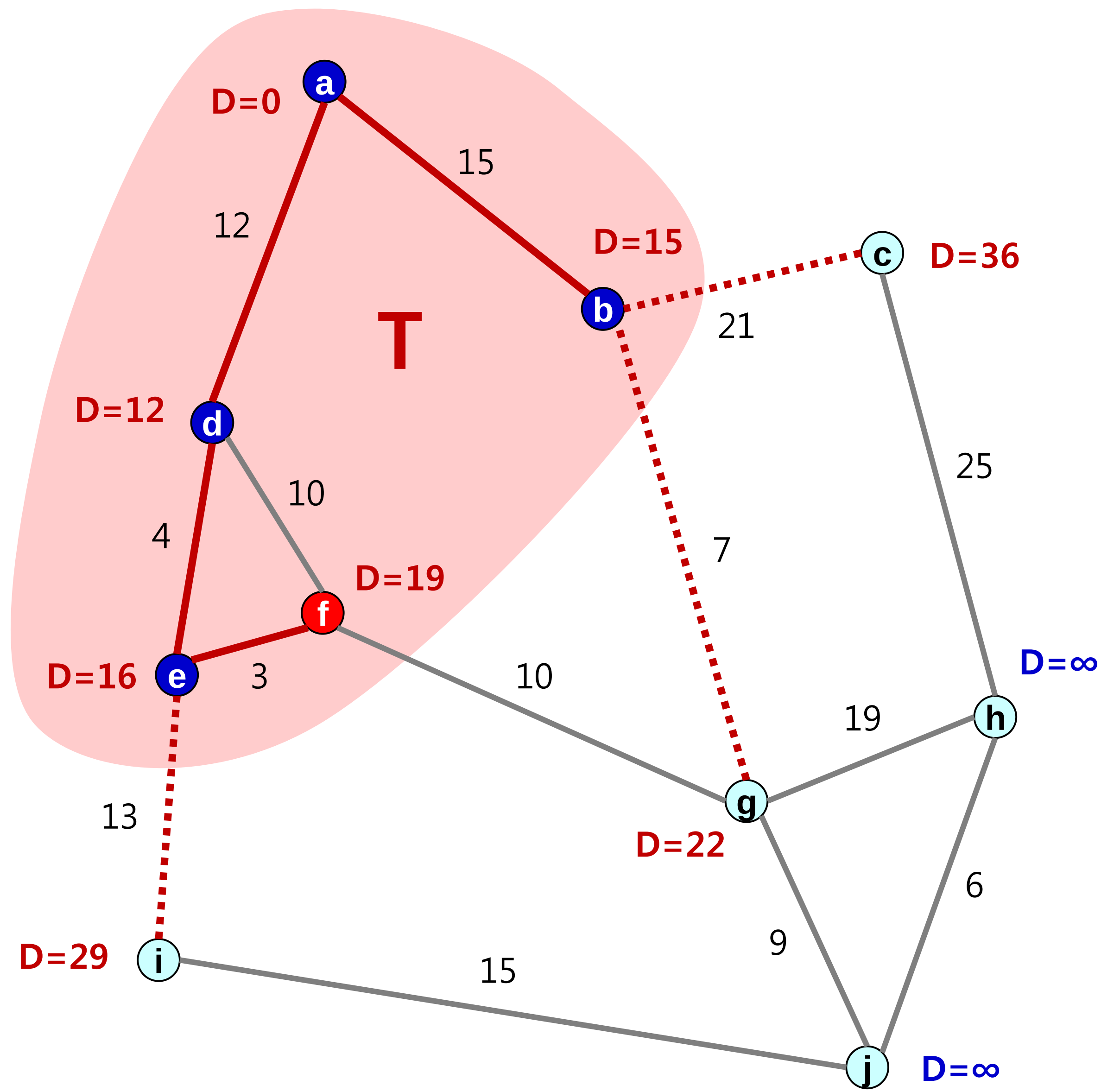

T
a
D=0
15
12
D=15
b
c
D=∞
21
D=12
d
25
10
4
7
D=12+10
=22
f
D=12+4
=16
e
3
10
D=∞
19
h
13
g
D=∞
6
9
D=∞
i
15
j
D=∞

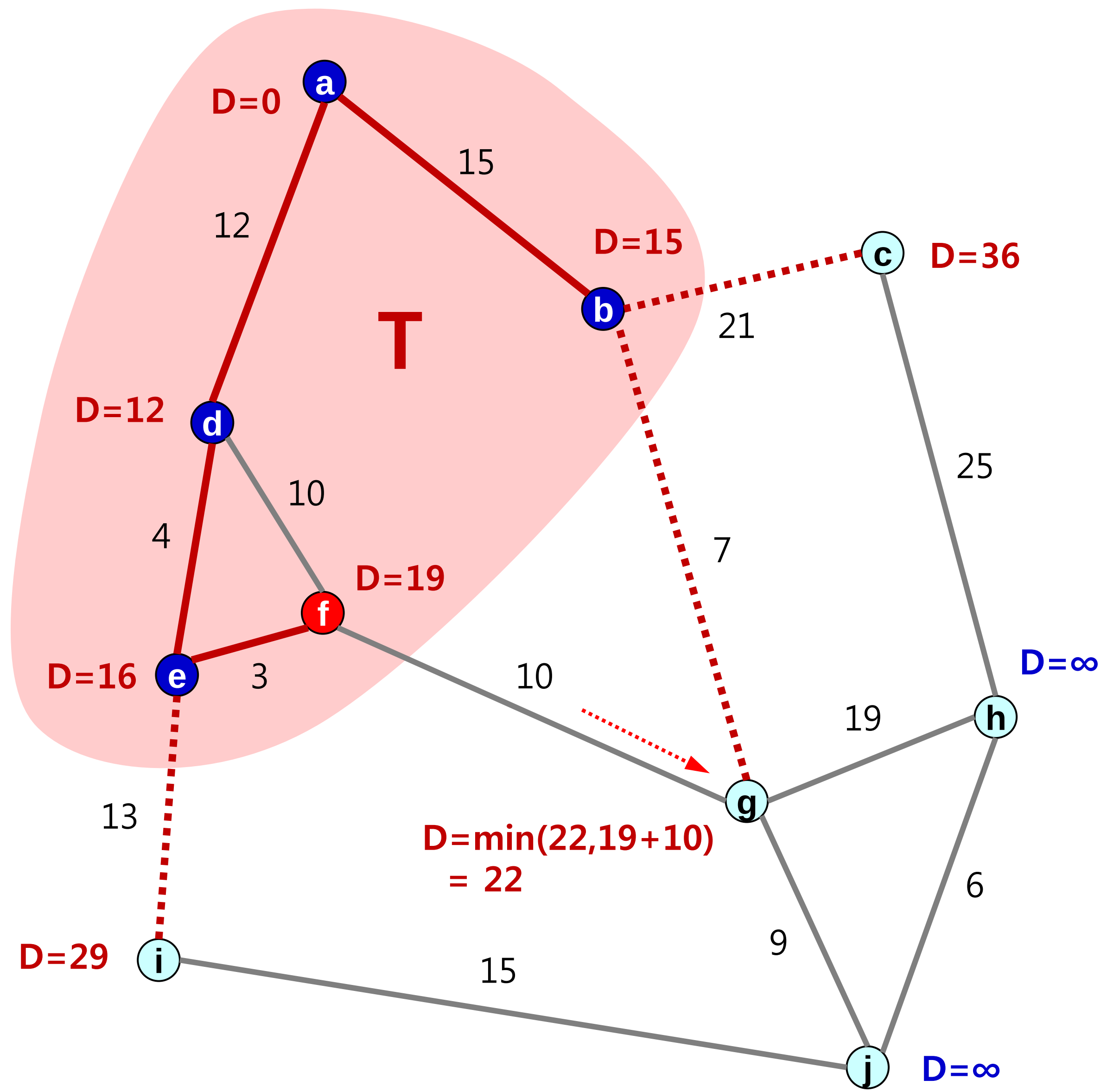

a
D=0
15
12
T
D=15
b
c
D=∞
21
D=12
d
25
10
4
7
D=22
f
D=16
e
3
10
D=∞
19
h
13
g
D=∞
6
9
D=∞
i
15
j
D=∞

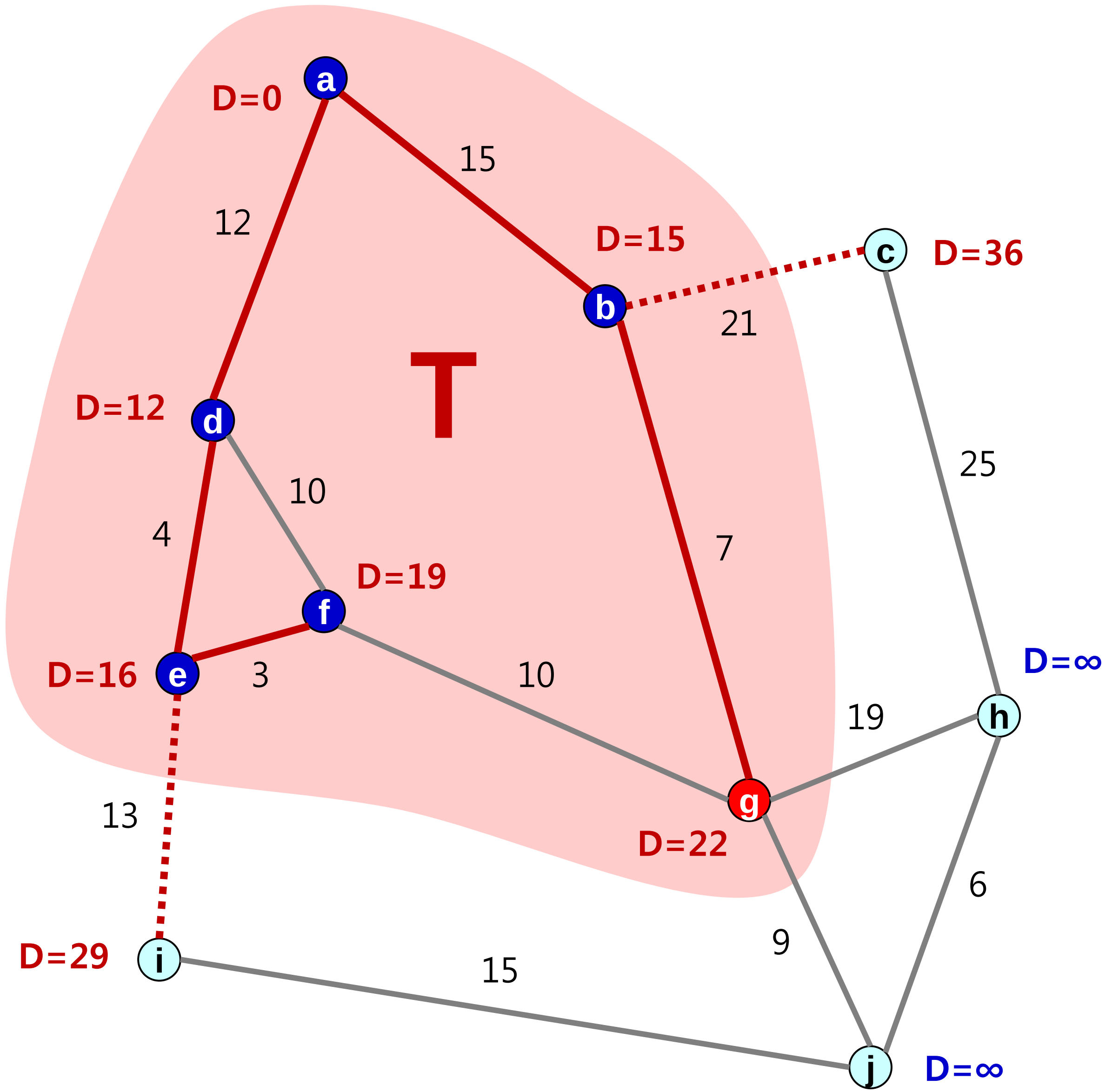

D=0
a
15
12
T
D=15
b
c
D=15+21
=36
21
D=12
d
25
10
4
7
D=22
f
D=16
e
3
10
D=∞
19
h
g
13
D=15+7
=22
6
9
D=∞
i
15
j
D=∞


a
b
c
d
e
f
g
h
i
j
T
D=0
D=15
D=36
D=12
D=22
D=16
D=22
D=∞
D=∞
D=∞
15
12
21
10
25
4
7
3
10
19
13
6
9
15

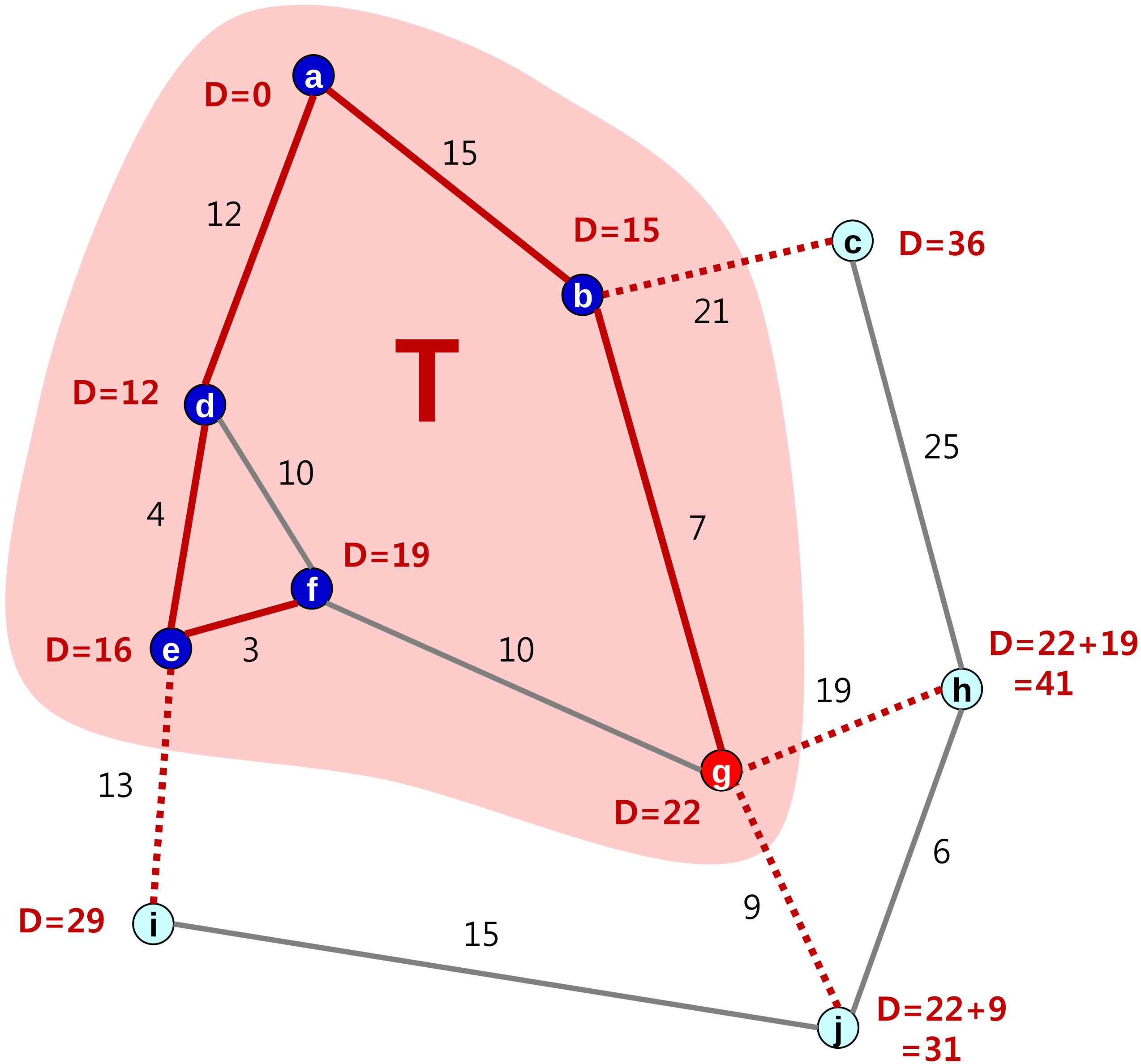

a
D=0
15
12
T
D=15
b
c
D=36
21
D=12
d
25
10
4
7
D=min(22, 16+3)
=19
f
D=16
e
3
10
D=∞
19
h
g
13
D=22
6
D=16+13
=29
i
9
15
j
D=∞

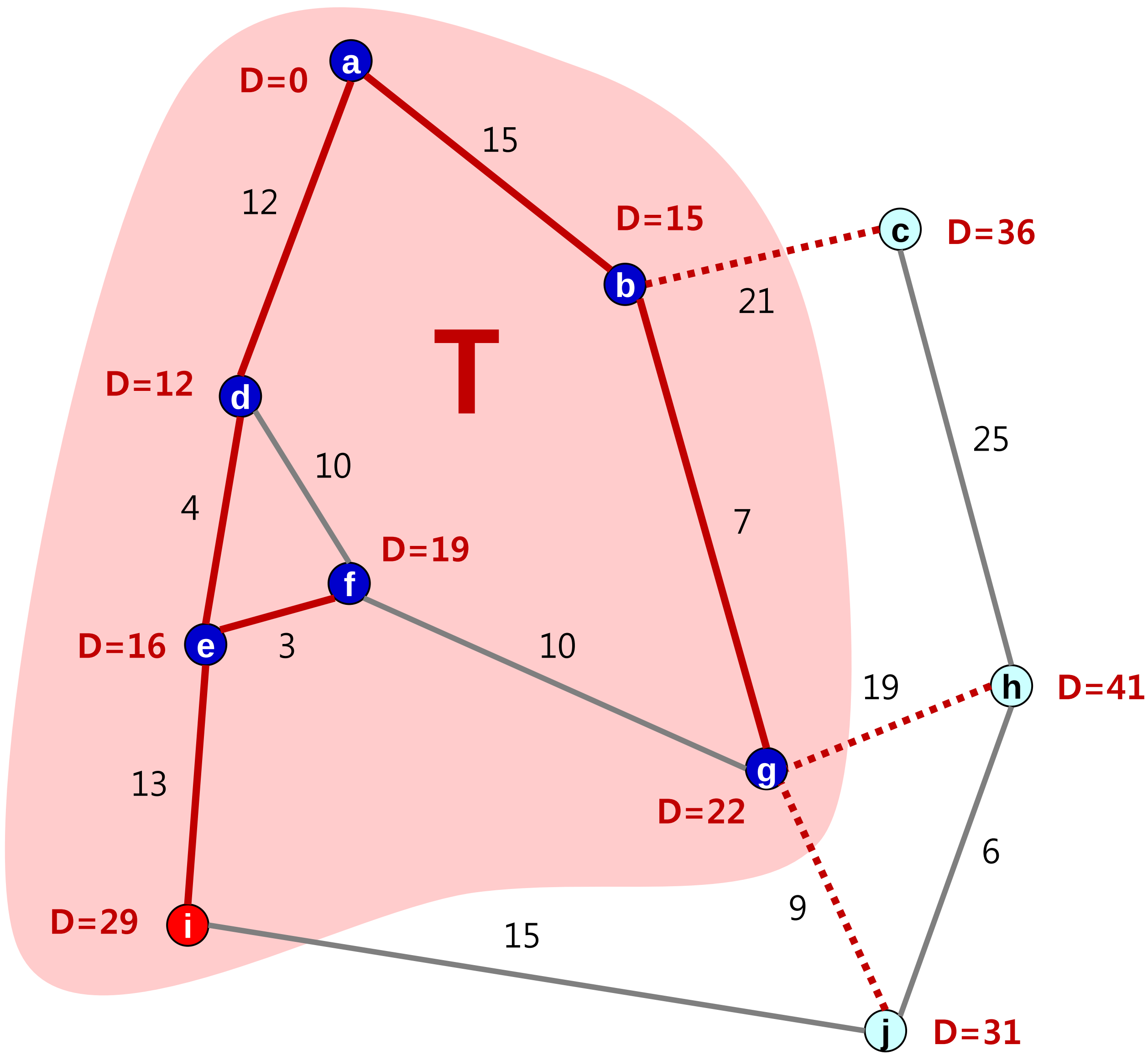

a
D=0
15
12
D=15
b
c
D=36
21
T
D=12
d
10
25
4
7
D=19
f
D=16
e
3
10
D=∞
19
h
13
g
D=22
6
9
D=29
i
15
j
D=∞

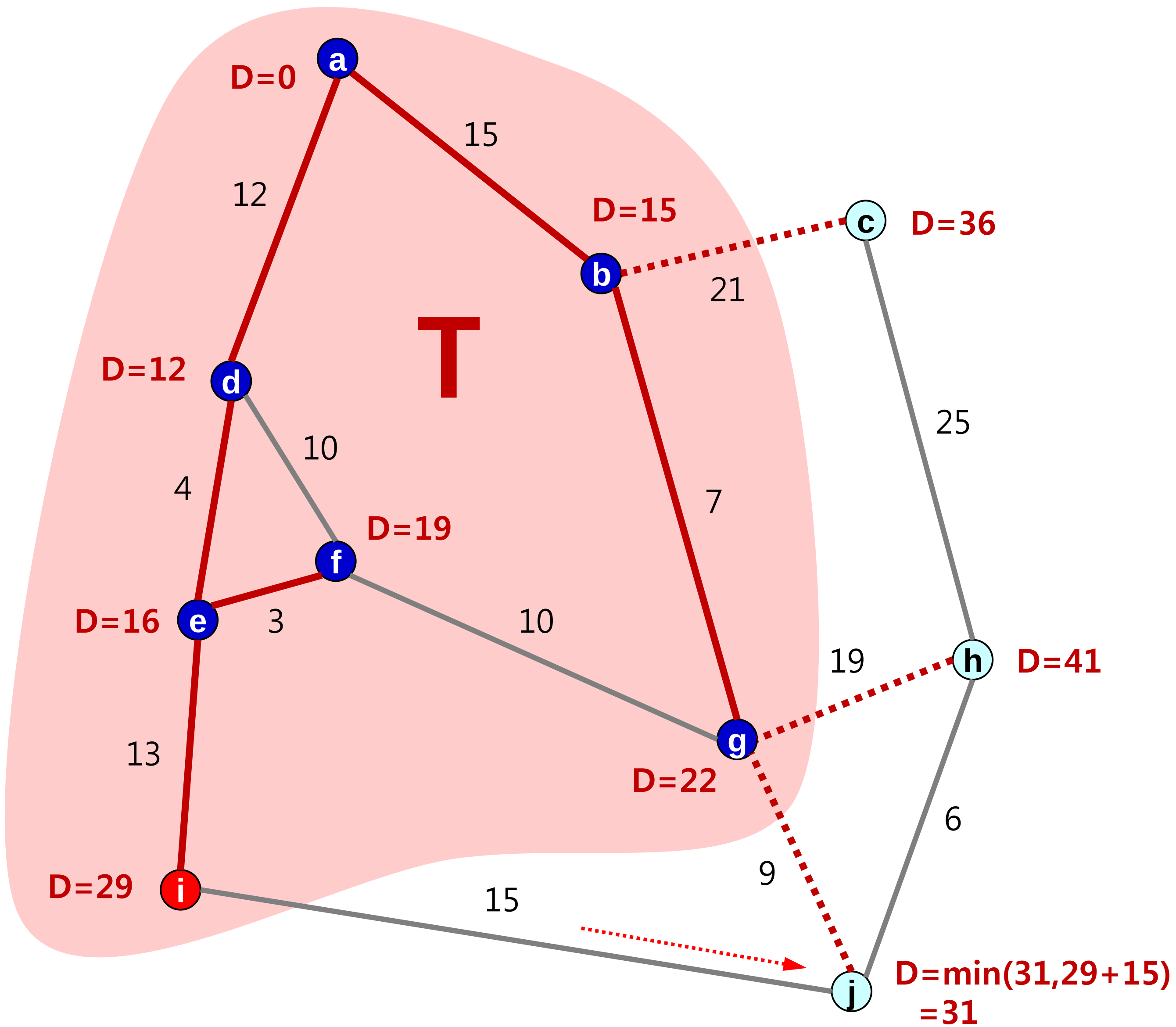

D=0
a
15
12
D=15
c
D=36
b
21
T
D=12
d
25
10
4
7
D=19
f
D=∞
D=16
e
3
10
19
h
g
13
D=min(22,19+10)
= 22
6
9
D=29
i
15
j
D=∞

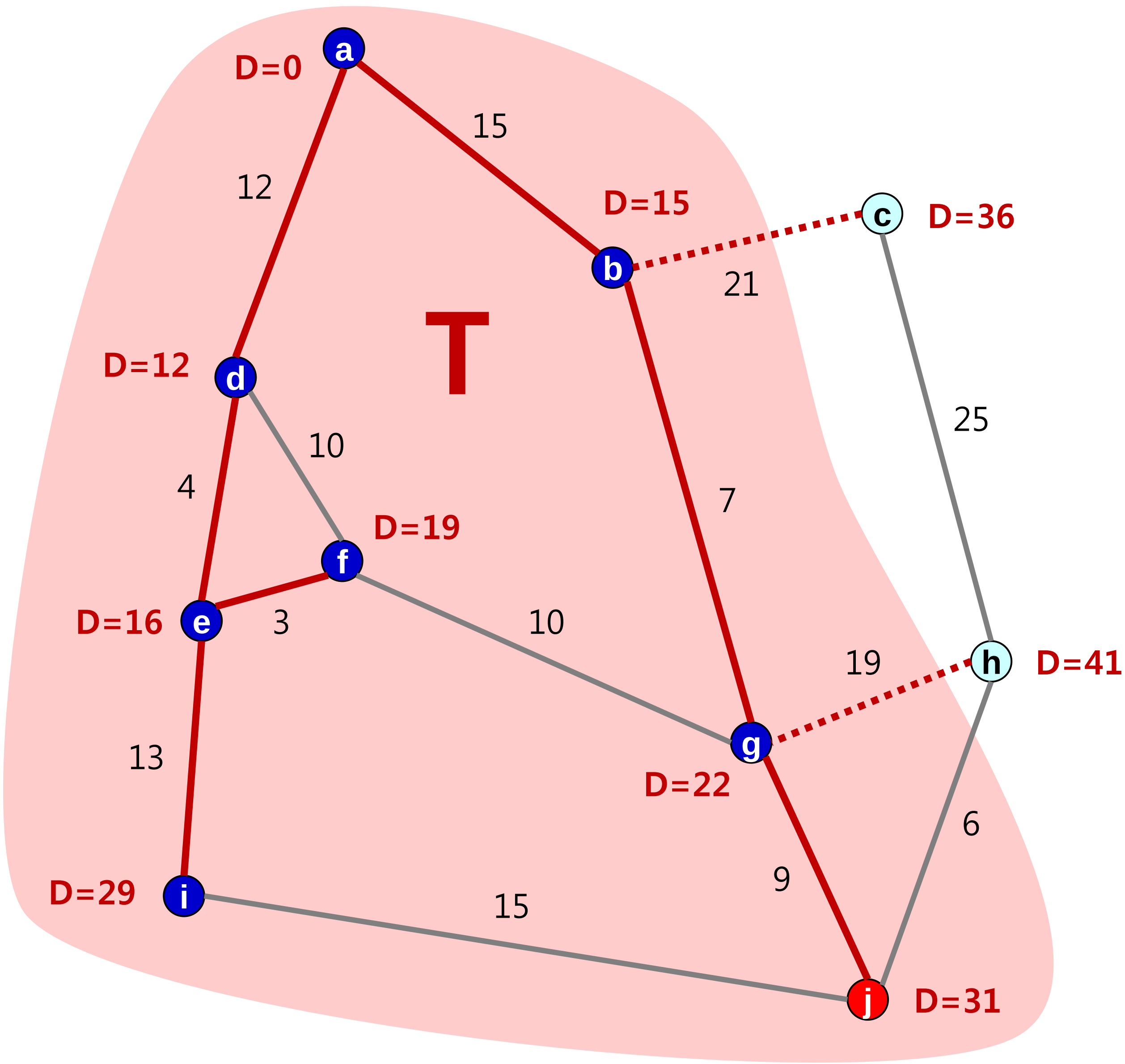


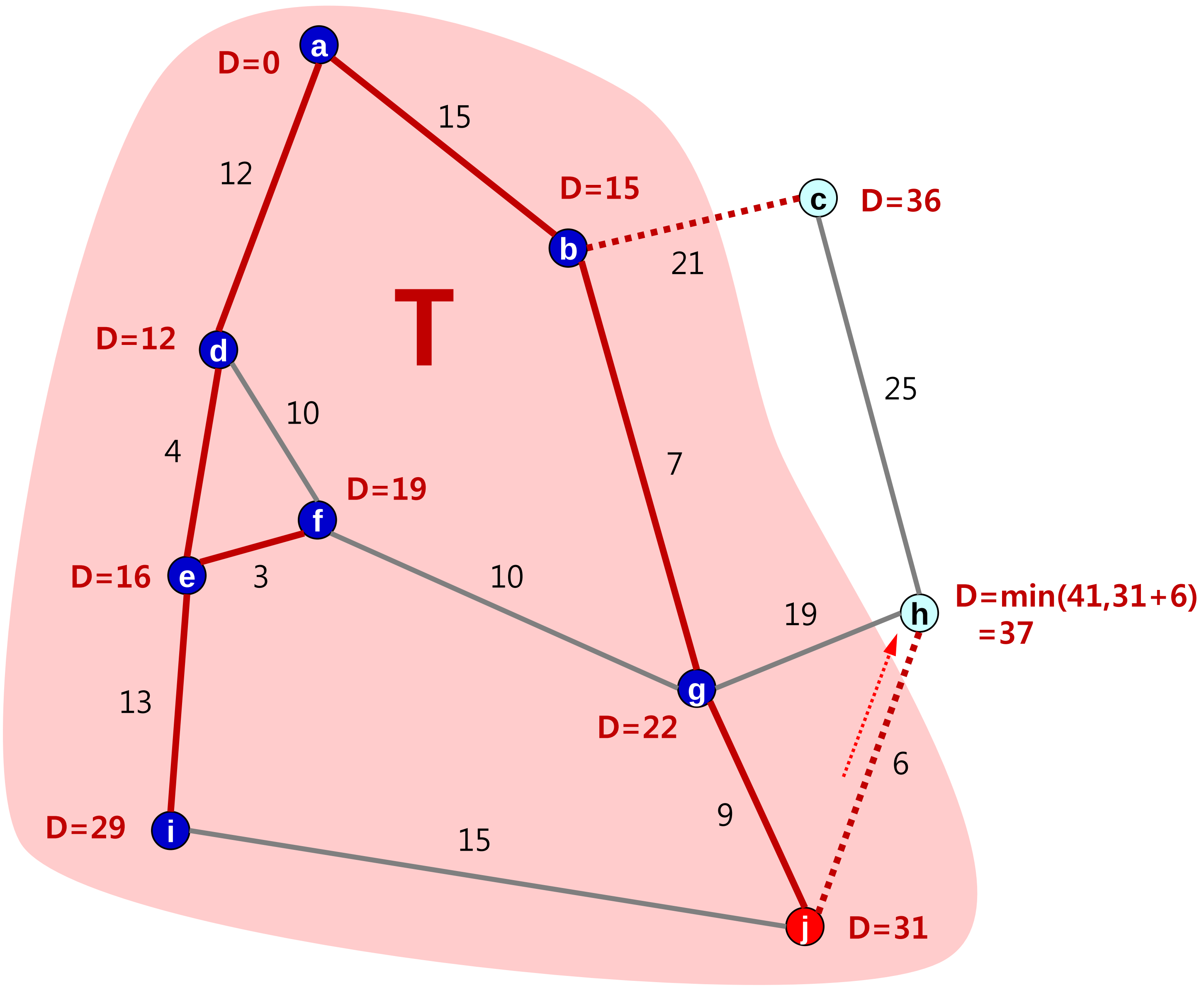
a
D=0
15
12
D=15
b
c
D=36
21
T
D=12
d
25
10
4
7
D=19
f
D=∞
e
D=16
3
10
19
h
g
13
D=22
6
9
D=29
i
15
j
D=∞


a
b
c
d
e
f
g
h
i
j
T
D=0
D=15
D=36
D=12
D=19
D=16
D=22
D=22+19
=41
D=29
D=22+9
=31
15
12
21
25
10
4
7
3
10
19
13
6
9
15

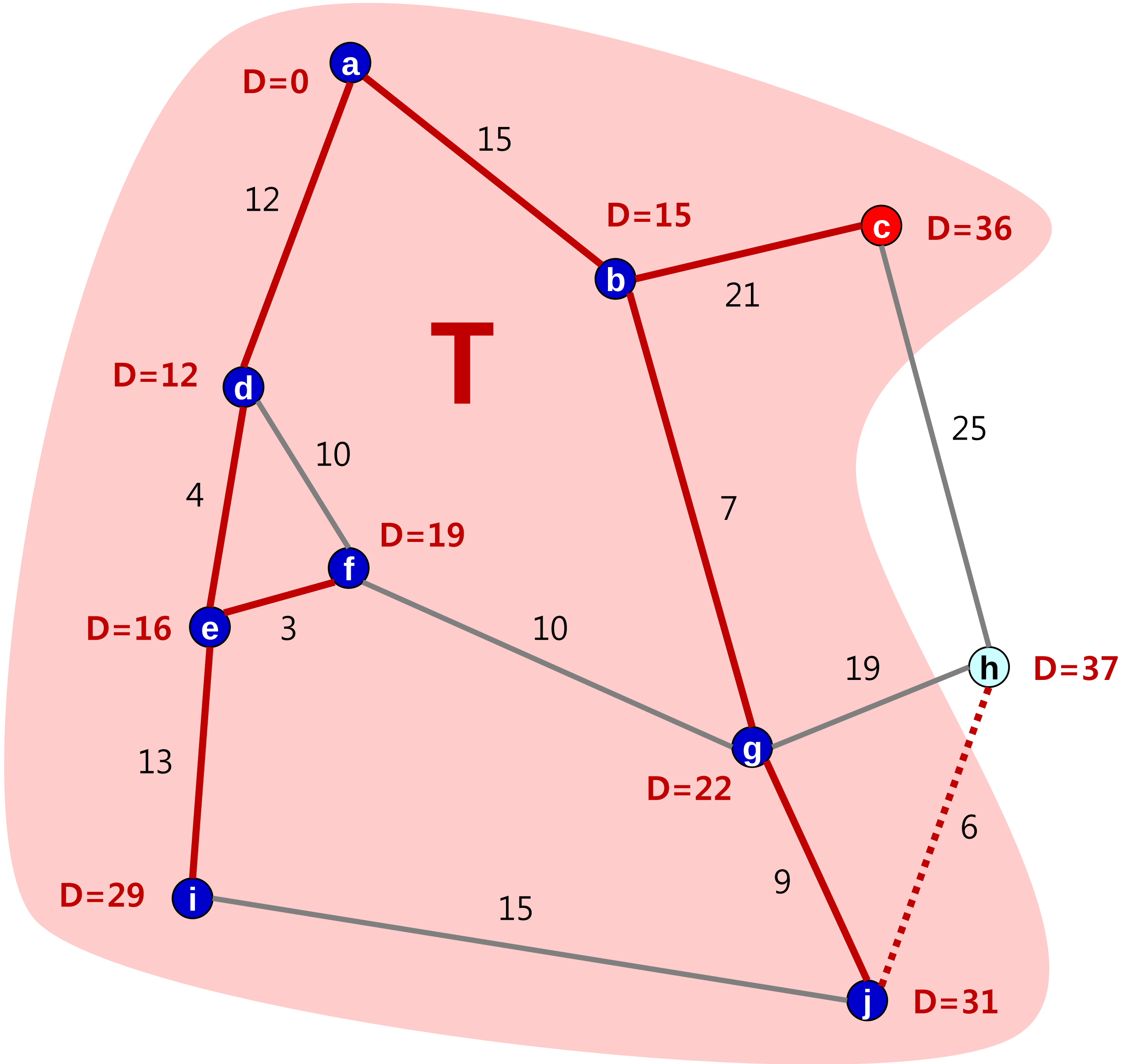

a
D=0
15
12
D=15
c
D=36
b
21
T
D=12
d
25
10
4
7
D=19
f
D=16
e
3
10
19
h
D=41
g
13
D=22
6
9
D=29
i
15
j
D=31

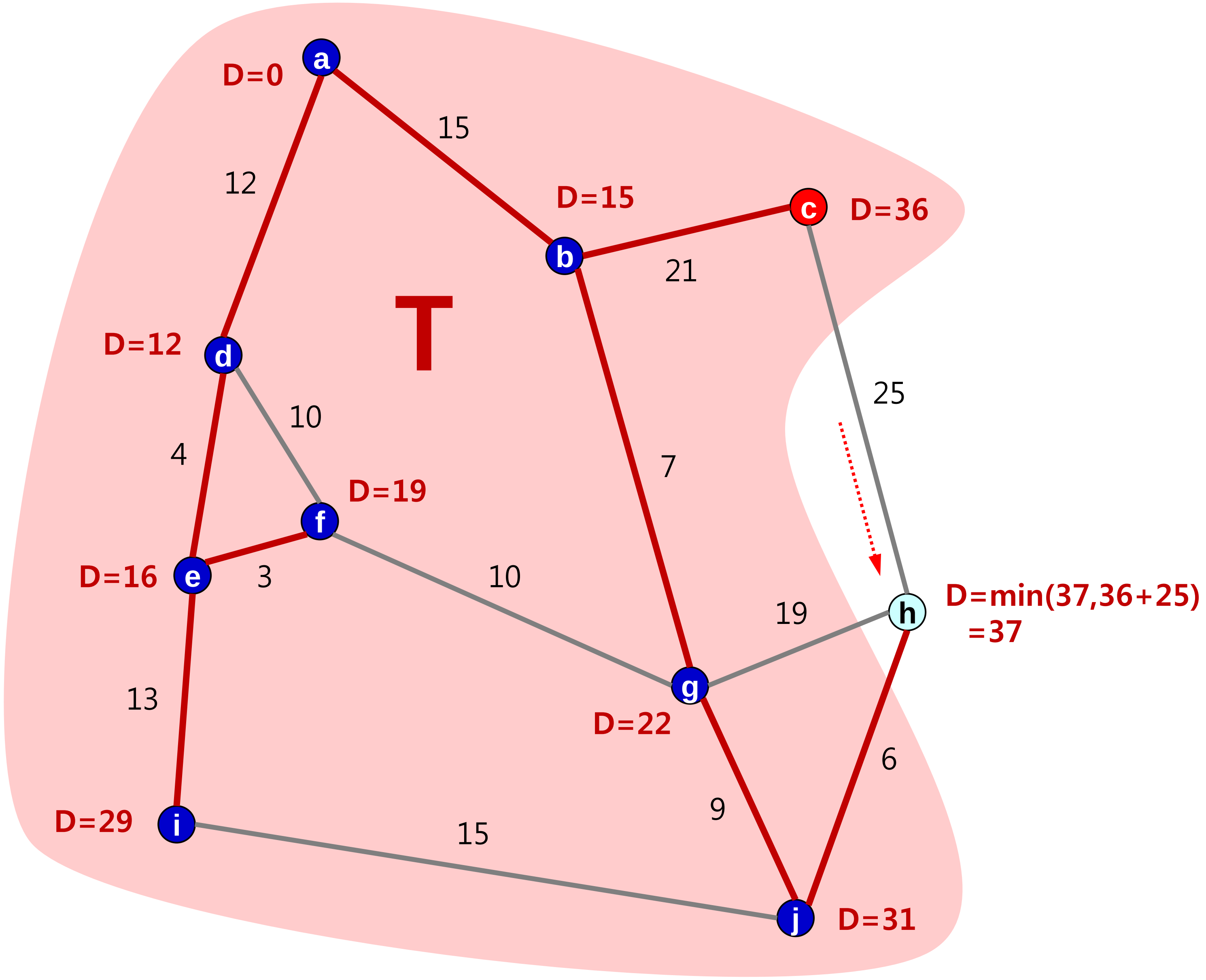

a
b
c
d
e
f
g
h
i
j
T
D=0
D=15
D=36
D=12
D=19
D=16
D=22
D=41
D=29
D=min(31,29+15)
=31
15
12
21
25
10
4
7
3
10
19
13
6
9
15

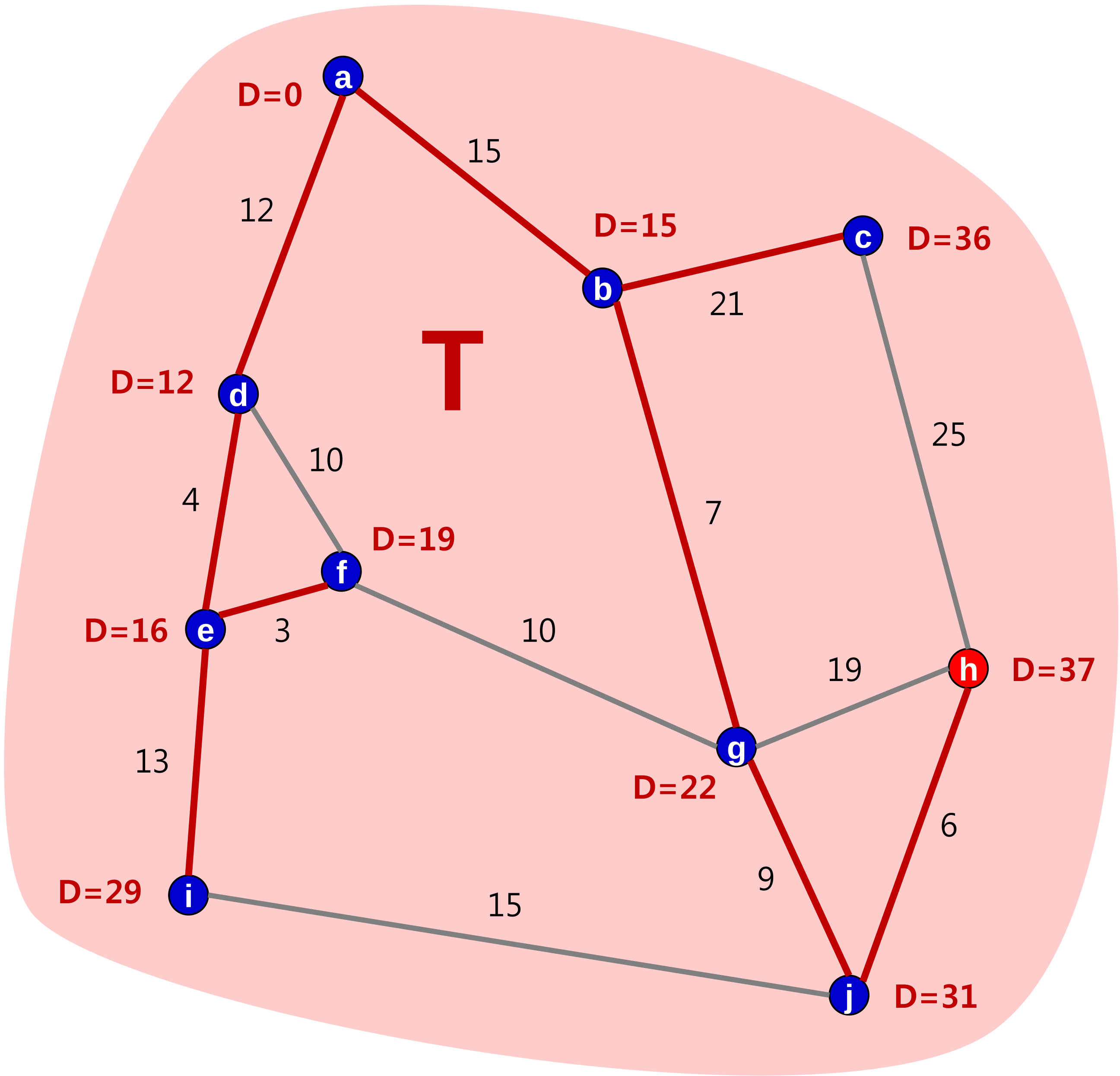

T
a
D=0
15
12
b
D=15
c
D=36
21
d
D=12
25
10
4
7
f
D=19
e
D=16
3
10
19
h
D=41
g
D=22
13
6
9
i
D=29
15
j
D=31

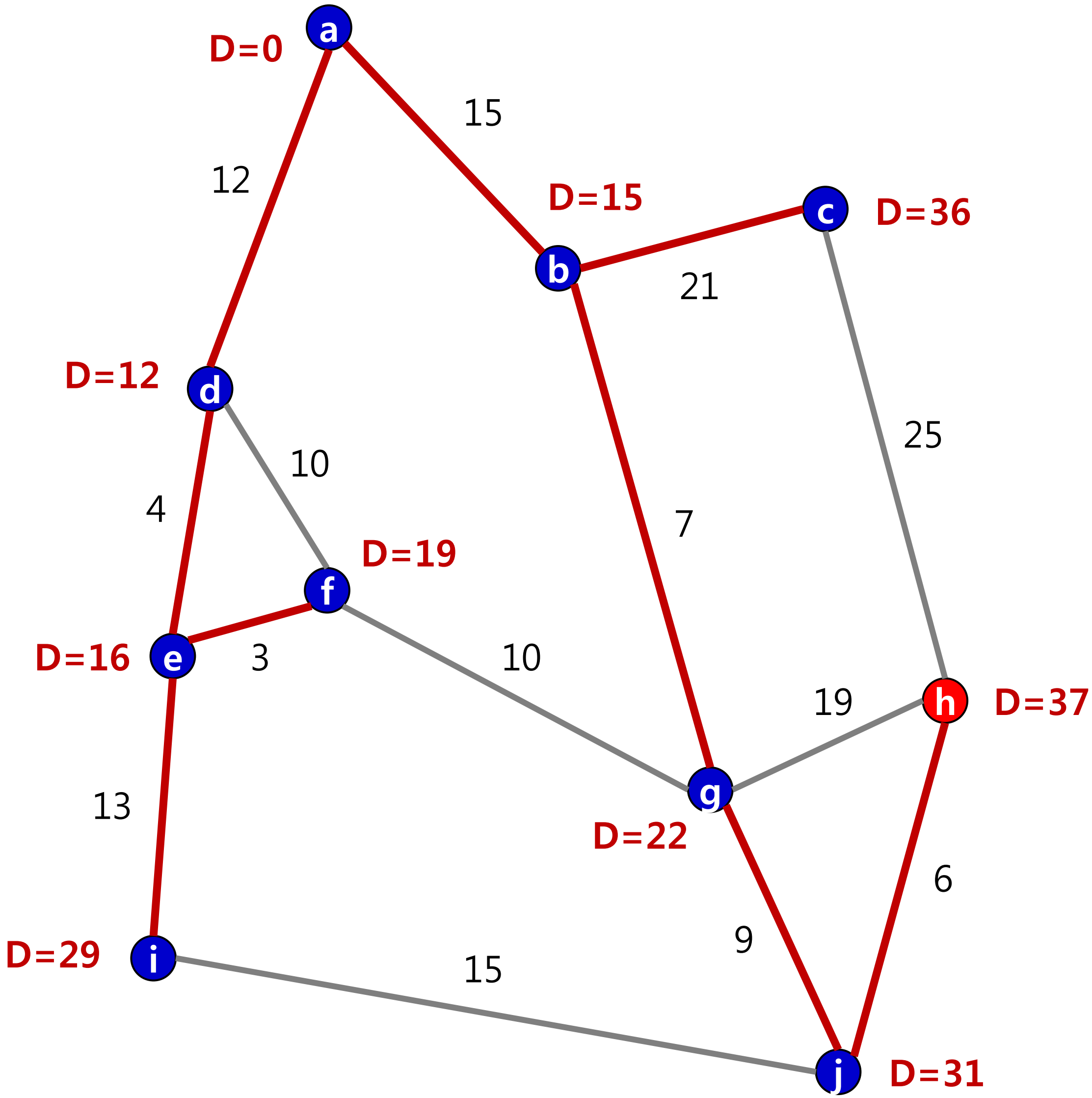


a
D=0
15
12
D=15
c
D=36
b
21
T
D=12
d
25
10
4
7
D=19
f
D=16
e
3
10
19
h
D=min(41,31+6)
=37
g
13
D=22
6
9
D=29
i
15
j
D=31


T
a
D=0
b
D=15
c
D=36
d
D=12
e
D=16
f
D=19
g
D=22
h
D=37
i
D=29
j
D=31
15
12
21
25
10
4
7
3
10
19
13
6
9
15


a
D=0
15
12
D=15
c
D=36
b
21
T
D=12
d
25
10
4
7
D=19
f
D=16
e
3
10
19
h
D=min(37,36+25)
=37
13
g
D=22
6
9
D=29
i
15
j
D=31


D=0
a
15
12
D=15
b
D=36
c
21
T
D=12
d
25
10
4
7
D=19
f
D=16
e
3
10
19
h
D=37
13
g
D=22
6
9
D=29
i
15
j
D=31



정점	거리	경로
a	0	a
b	15	a-b
c	36	a-b-c
d	12	a-d
e	16	a-d-e
f	19	a-d-e-f
g	22	a-b-g
h	37	a-b-g-j-h
i	29	a-d-e-i
j	31	a-b-g-j

시간복잡도

- $while$ 루프가 $(n-1)$번 반복된다.
- line 3에서 v_{min}에 연결된 점 w의 수가 최대 $(n-1)$개이므로, 각 $D[w]$를 갱신하는데 걸리는 시간은 $O(n)$이다.
- line 4에서 v_{min}을 찾는데 $O(n)$ 시간이 걸린다. 왜냐하면 배열 D에서 최소값을 찾는 것이기 때문이다.
- 따라서 시간복잡도는 $(n-1) \times \{O(n) + O(n)\} = O(n^2)$이다.

응용 분야

- 맵퀘스트(Mapquest)와 구글(Google) 웹사이트의 지도 서비스
- 자동차 네비게이션을 비롯한 교통 공학
- 네트워크와 통신 분야
- 경영 공학의 운영 연구 (Operation Research)
- 로봇 공학
- VLSI 디자인 분야 등

7.6 여행자 문제

- 여행자 문제(Traveling Salesman Problem, TSP)
  - n개의 도시와 각 도시간의 거리가 주어졌을 때,
    각 도시를 한번씩만 방문하면서 모든 도시를 방문하는
    최단경로를 구하는 문제

- 여행자 문제의 조건
  - 대칭성: 도시 A에서 B로 가는 거리는 B에서 A로 가는 거리와 같다.
  - 삼각 부등식 특성: 인접한 도시 A에서 도시 B로 가는 거리는
    도시 A에서 다른 도시 C를 경유하여 도시 B로 가는 거리보다 짧다.

백트래킹 기법

- 백트래킹(Backtracking) 기법
  - 해를 찾는 도중에 '막히면' (즉, 해가 아니면) 되돌아가서 다시 해를 찾아 가는 기법
- 백트래킹 기법은 최적화(optimization) 문제와 결정(decision) 문제를 해결할 수 있음
  - 결정 문제: 문제의 조건을 만족하는 해가 존재하는지의 여부를 'yes' 또는 'no'로 답하는 문제
    - 미로 찾기: 출구가 있는가?
    - 해밀토니안 사이클 (Hamiltonian Cycle) 문제
    - 부분 집합의 합(Subset Sum) 문제 등

여행자 문제(TSP)를 위한 백트래킹 알고리즘

```
tour = [시작점]                // tour는 점의 순서
bestSolution = (tour, ∞)      // tour가 시작점만 있으므로 현재 거리는 ∞

BacktrackTSP(tour)
if (tour가 완전한 해이면) {
    if (tour의 거리 < bestSolution의 거리)     // 더 짧은 해를 찾았으면
        bestSolution = (tour, tour의 거리)
} else {
    for (tour를 확장 가능한 각 점 v에 대해서) {
        newTour = tour + v                    // 기존 tour의 뒤에  v 를 추가
        if (newTour의 거리 < bestSolution의 거리)
            BacktrackTSP(newTour)
    }
}
```

BacktrackTSP 알고리즘 해설

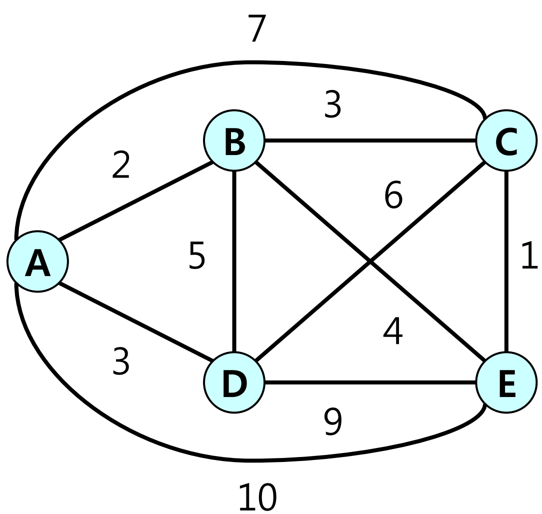
- tour
  - 여행자가 다니는 경로상의 점의 순서 (sequence)
- bestSolution
  - 현재까지 찾은 가장 우수한(거리가 짧은) 해로서
    2개의 성분 (tour, tour의 거리)으로 나타낸다.
  - bestSolution의 tour의 거리는 'bestSolution의 거리'로 표현
- 변수 초기값
  - tour = [시작점]   // tour는 점의 순서
  - bestSolution = (tour, ∞)   // 현재 거리의 초기값은 ∞

- Line 1~3:
  - 현재 tour가 완전한 해이면서 현재까지 찾은 가장 우수한 해인 bestSolution의 거리보다 짧으면 현재 tour로 bestSolution이 갱신
- Line 4~8:
  - tour가 아직 완전한 해가 아닐 때 수행
- Line 5:
  - for 루프: 현재 tour에서 확장 가능한 각 점에 대해서 루프 내부 수행
  - 확장 가능한 점: tour에 없는 점으로서 tour의 마지막 점과 연결된 점
- Line 6:
  - tour를 확장(확장 가능한 점을 기존 tour에 추가)하여 newTour를 얻음

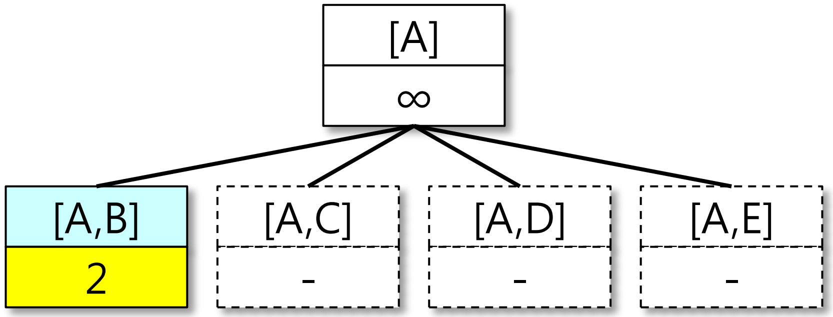
- Line 7~8:
  - 확장된 newTour의 거리가 bestSolution의 거리보다 짧으면, newTour에 대해 알고리즘을 재귀 호출한다.
  - 만일 newTour의 거리가 bestSolution의 거리보다 같거나 길면, newTour를 확장하여도 현재까지의 bestSolution의 거리보다 짧은 tour를 얻을 수 없기 때문에 가지치기(pruning)를 한다.
  - 가지를 친 경우에는 다음의 확장 가능한 점에 대해서 루프가 수행

BacktrackTSP 알고리즘 수행과정

- 시작점이 A이므로, tour=[A]이고, bestSolution=([A],∞)이다.
- BacktrackTSP(tour)를 호출하여 해 탐색 시작

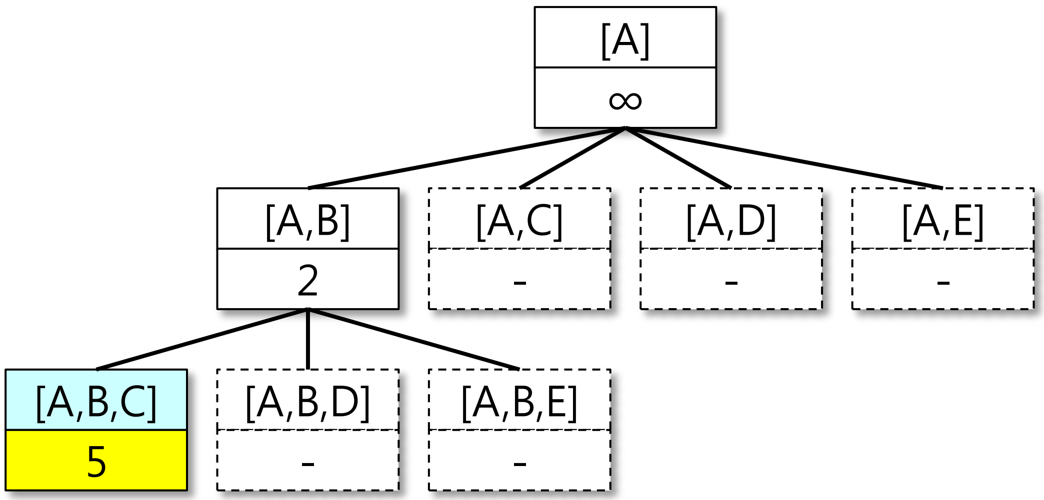


- Line 1:
  - [A]가 완전한 해가 아니므로 line 5의 for 루프 수행
- Line 5:
  - for 루프에서 tour [A]를 확장할 수 있는 점은 점 B, C, D, E가 있다.
  - 먼저 점 B에 대해서 line 6~8이 수행된다.
- Line 6:
  - newTour=[A,B], newTour의 거리는 2가 된다.
  - 여기서 2=선분(A,B)의 가중치 2



- Line 7~8:
  - newTour의 거리 2가 bestSolution의 거리 ∞보다 짧으므로, BacktrackTSP([A,B])를 재귀 호출 한다. 확장 가능한 점 C, D, E에 대해서는 BacktrackTSP([A,B]) 호출을 다 마친 후에 각각 수행한다.
- Line 1:
  - BacktrackTSP([A,B])가 호출되면, [A,B]가 완전한 해가 아니므로 line 5의 for 루프 수행
- Line 5:
  - for 루프에서 tour [A,B]를 확장할 수 있는 점은 점 C, D, E가 있다.
  - 먼저 점 C에 대해서 line 6~8이 수행된다.

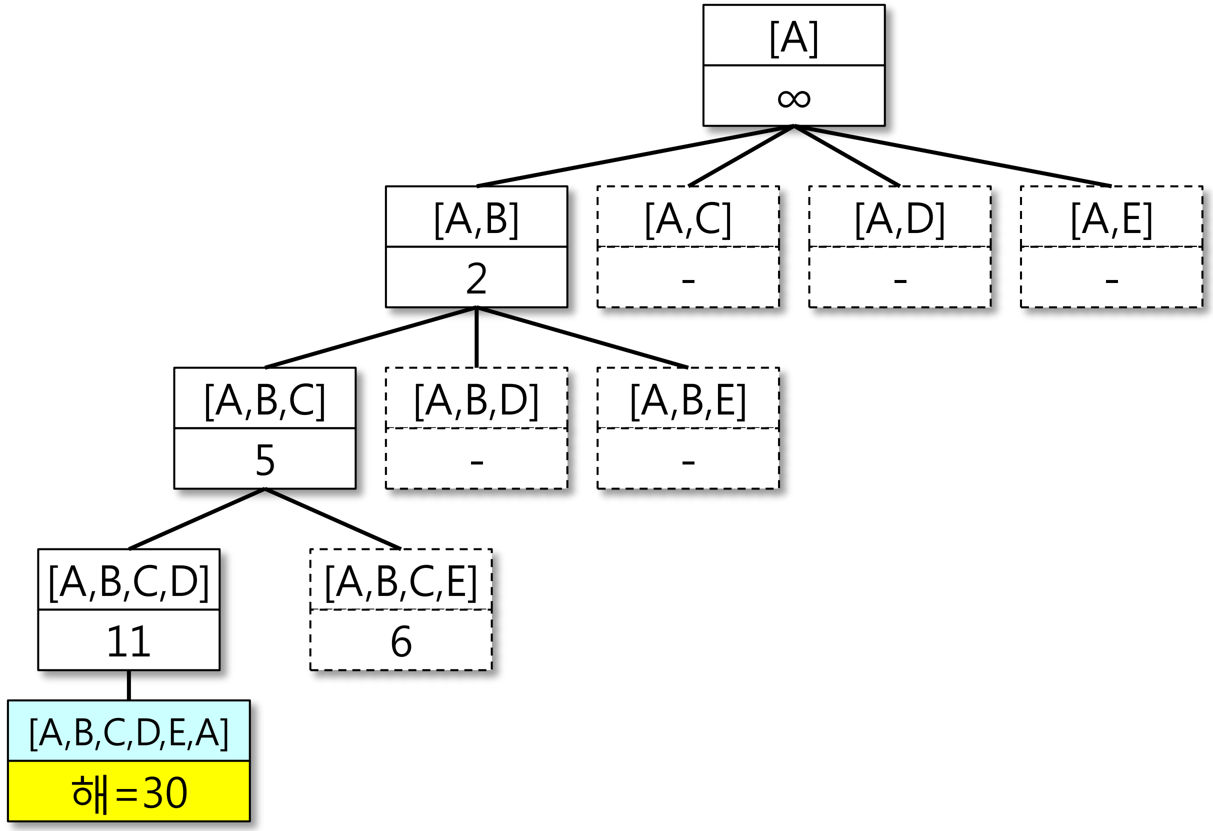
- Line 6:
  - newTour=[A,B,C]가 되고, newTour의 거리는 5.
  - 여기서 5=tour[A,B]의 거리 2 + 선분(B,C)의 가중치 3



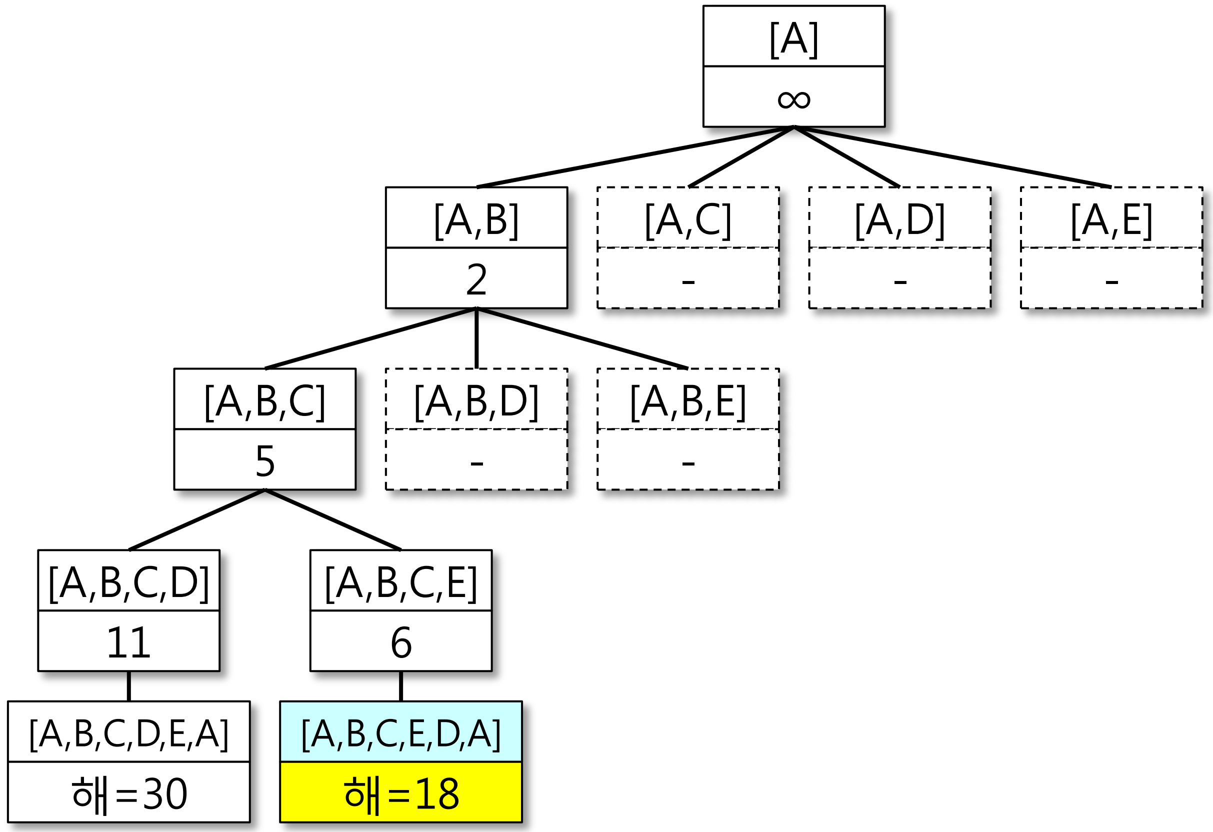
- Line 7~8:
  - newTour의 거리 5가 bestSolution의 거리 ∞보다 짧으므로, BacktrackTSP([A,B,C])를 재귀 호출 한다. 확장 가능한 점 D, E에 대해서는 BacktrackTSP([A,B,C]) 호출을 다 마친 후에 각각 수행한다.

    . . .

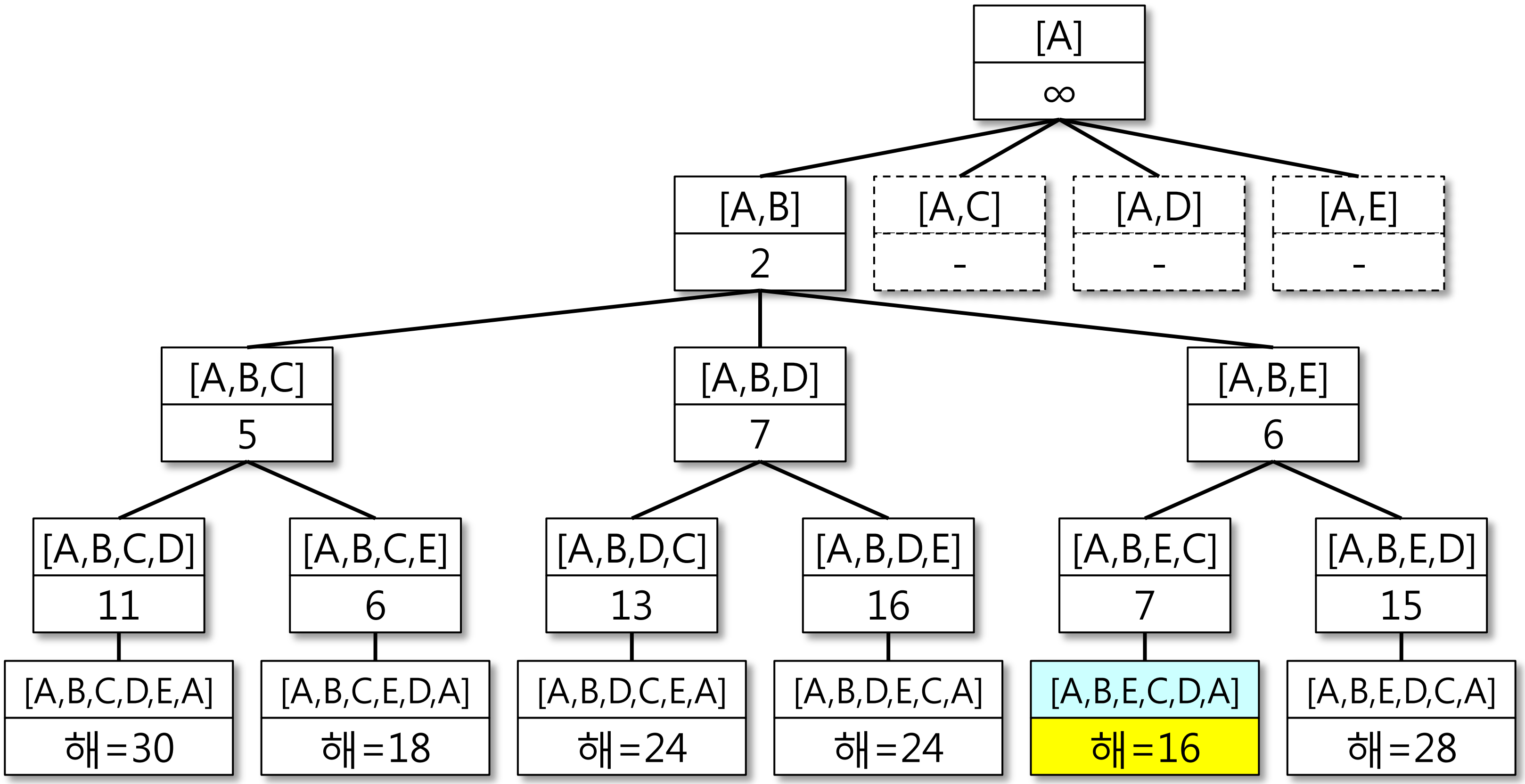
- 이와 같이 탐색을 계속하면, 다음과 같이 첫 번째 완전한 해를 찾는다. 이때 bestSolution=([A,B,C,D,E,A], 30)이 된다.


[A]
∞
[A,B]
2
[A,C]
-
[A,D]
-
[A,E]
-
[A,B,C]
5
[A,B,D]
-
[A,B,E]
-
[A,B,C,D]
11
[A,B,C,E]
6
[A,B,C,D,E,A]
해=30

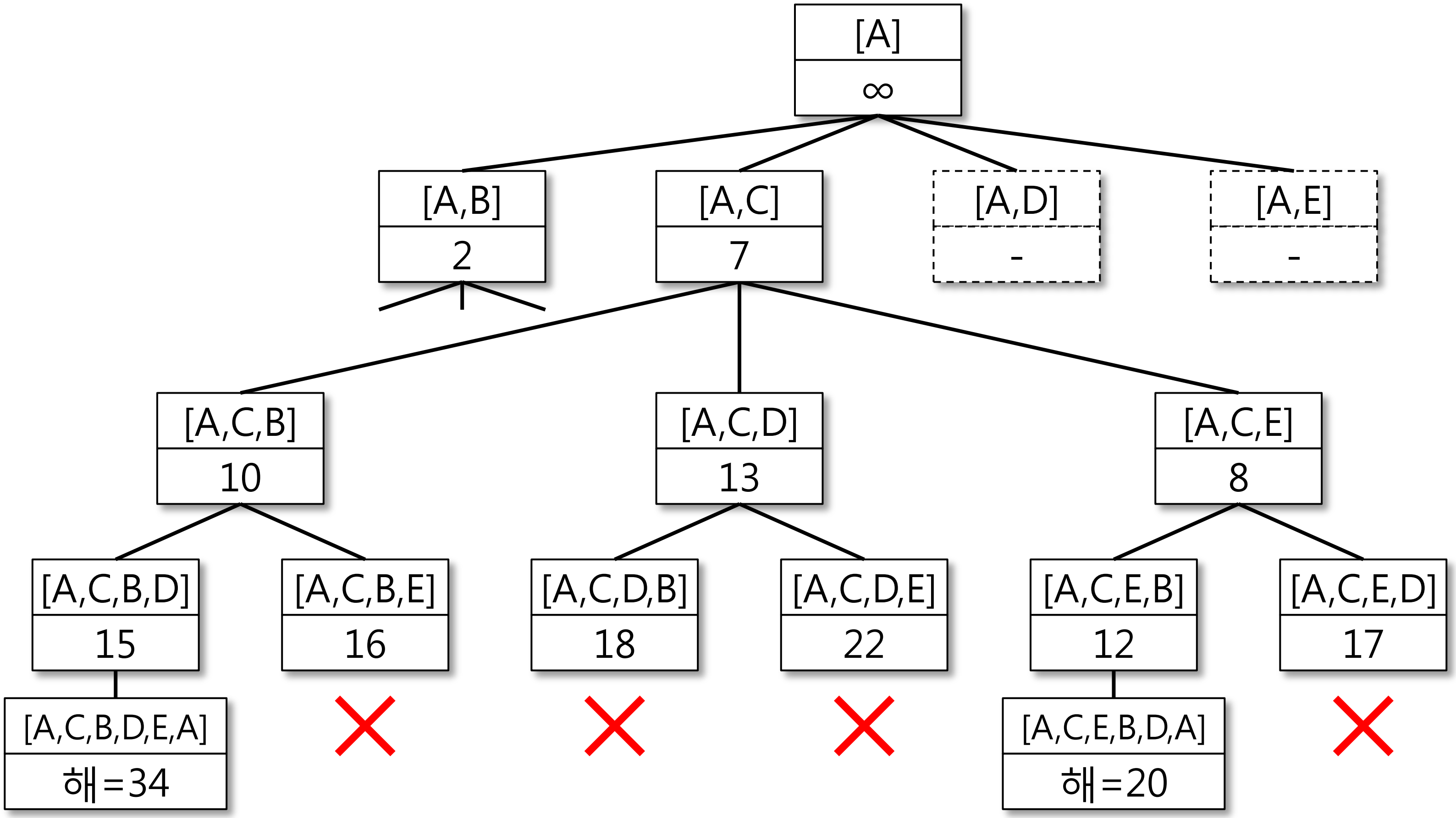
- 첫 번째 완전한 해를 찾은 후에는 다음과 같이 수행되며, 이때 더 짧은 해를 찾으므로 bestSolution=([A,B,C,E,D,A], 18)이 된다.



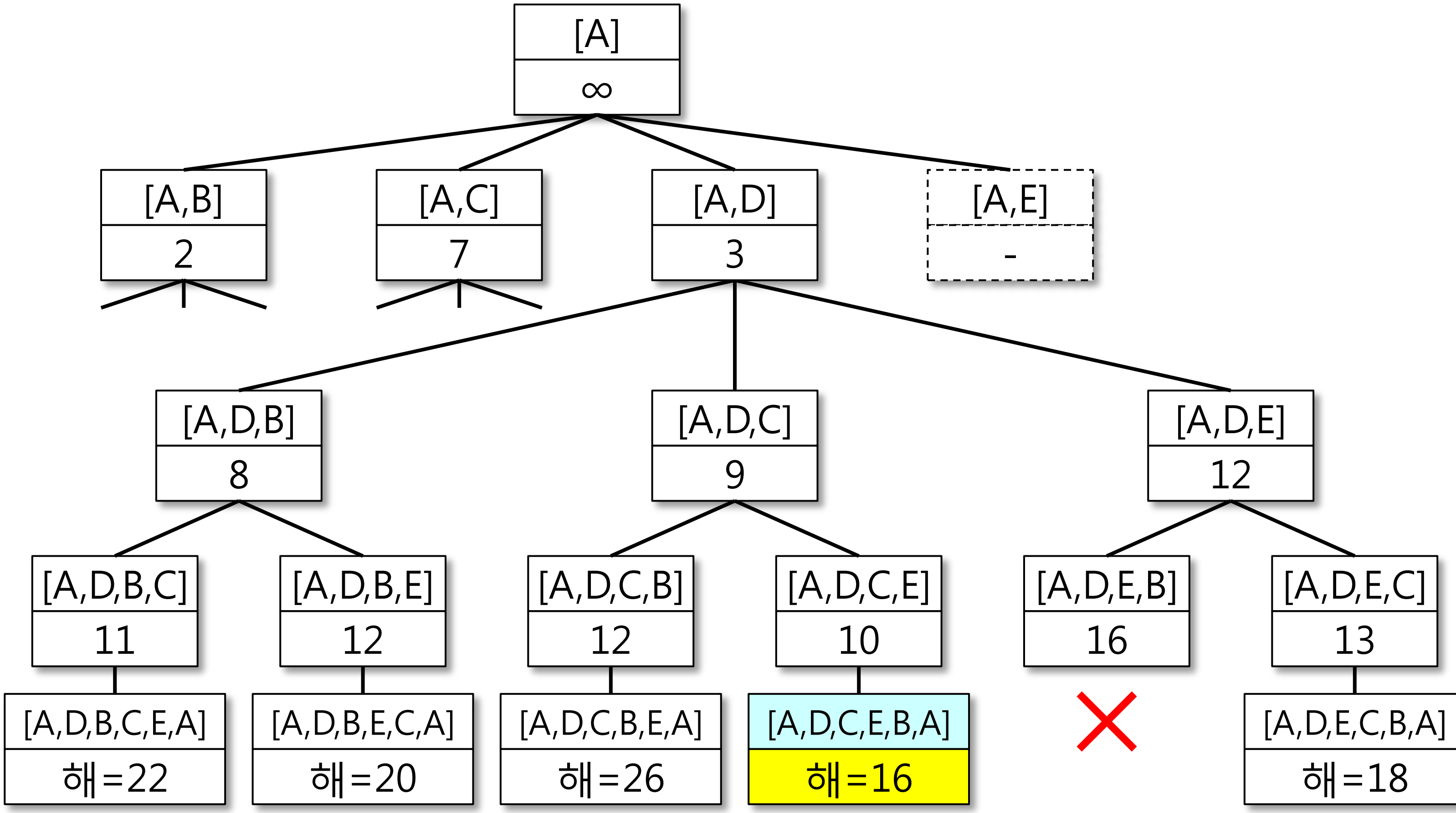
- 다음은 tour=[A,B]에 대해서 모든 수행을 마친 결과이다.
- 현재 bestSolution=([A,B,E,C,D,A], 16)이다.



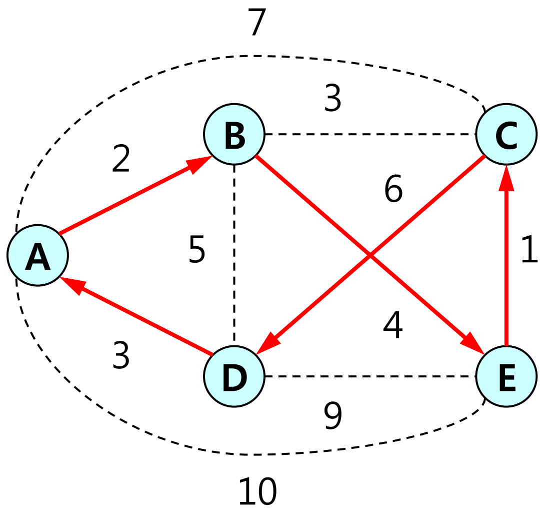
- 다음은 tour=[A,C]에 대해서 모든 수행을 마친 결과이다.
- 그러나 bestSolution ([A,B,E,C,D,A], 16) 보다 더 우수한 해는 탐색되지 않았고, ×로 표시된 4개의 상태는 각각 bestSolution의 거리보다 짧지 않아서 가지치기된 것이다.


[A]
∞
[A,B]
2
[A,C]
7
[A,D]
-
[A,E]
-
[A,C,B]
10
[A,C,D]
13
[A,C,E]
8
[A,C,B,D]
15
[A,C,B,E]
16
[A,C,D,B]
18
[A,C,D,E]
22
[A,C,E,B]
12
[A,C,E,D]
17
[A,C,B,D,E,A]
해=34
[A,C,E,B,D,A]
해=20

- 다음은 tour=[A,D]에 대해서 모든 수행을 마친 결과이다.
- 이때 bestSolution보다 우수한 해는 탐색되지 않았으나 같은 거리의 해를 찾는데 이 해는 bestSolution tour의 역순이었다.
- 역시 ×로 표시된 1개의 상태는 bestSolution의 거리보다 짧지 않아서 가지치기된 것이다.

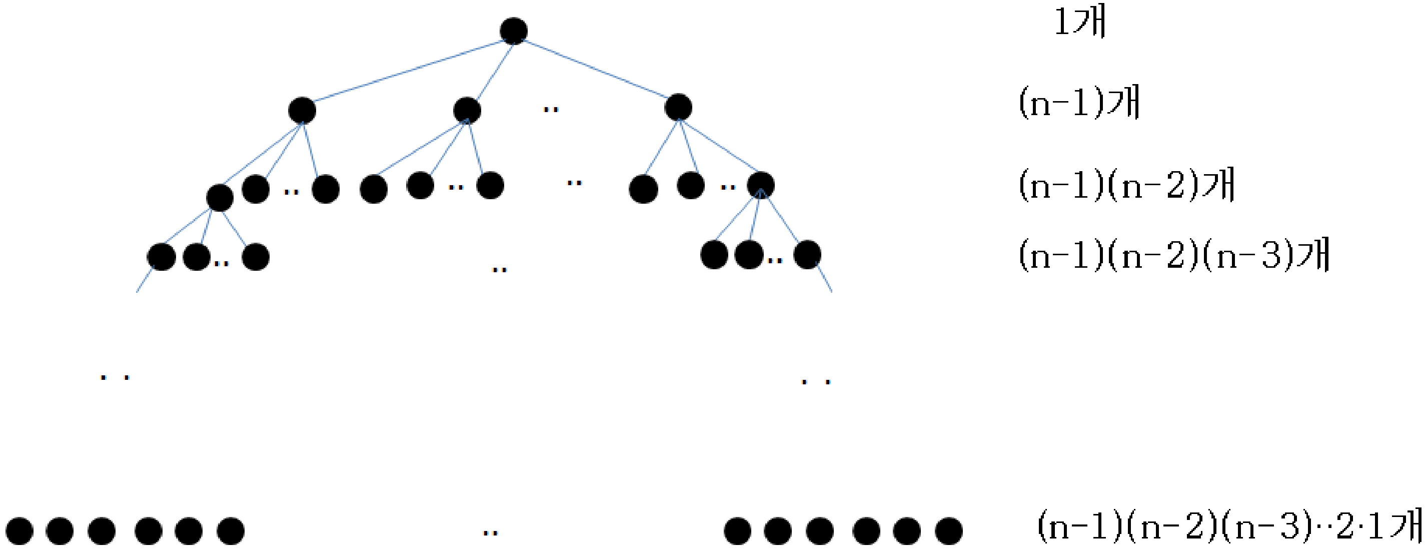

[A]
∞
[A,B]
2
[A,C]
7
[A,D]
3
[A,E]
-
[A,D,B]
8
[A,D,C]
9
[A,D,E]
12
[A,D,B,C]
11
[A,D,B,E]
12
[A,D,C,B]
12
[A,D,C,E]
10
[A,D,E,B]
16
[A,D,E,C]
13
[A,D,B,C,E,A]
해=22
[A,D,B,E,C,A]
해=20
[A,D,C,B,E,A]
해=26
[A,D,C,E,B,A]
해=16
[A,D,E,C,B,A]
해=18

- 마지막으로 tour=[A,D]에 대해서 탐색을 수행하여도 bestSolution보다 더 우수한 해는 발견되지 않았다.
- 따라서 최종해=[A,B,E,C,D,A]이고, 거리=16이다.



시간복잡도

- 백트래킹 알고리즘의 시간복잡도는
  상태 공간 트리(state space tree)의 노드 수에 비례한다.
- n개의 점이 있는 입력 그래프에 대해서 BacktrackTSP 알고리즘이 탐색하는 최대 크기의 상태 공간 트리는 다음과 같다.



- 앞의 트리의 단말 노드 수만 계산해도 $(n-1)!$개가 되므로 팩토리얼형 시간복잡도인 $O(n!)$가 된다.
- 문제에 따라 이진트리 형태의 상태 공간 트리가 형성되는데, 이때에도 최악의 경우 2^n개의 노드를 모두 탐색해야 하므로 지수형 시간복잡도인 $O(2^n)$이 걸리게 된다.
- 위의 경우는 모든 경우를 다 검사하여 해를 찾는 완결탐색(Exhaustive Search)의 시간복잡도와 같다.
- 일반적으로 백트래킹 기법은 '가지치기'를 하기 때문에 완결탐색보다 훨씬 효율적으로 동작한다.

P-문제와 NP-완전문제

- P(Polynomial time)-문제
  - 다항식 시간복잡도 $O(n^k)$ 이내의 쉬운 문제
  - 자료구조, 알고리즘에서 다루는 대부분의 문제
- NP-완전(Nondeterministic Polynomial Complete) 문제
  - 지수시간 시간복잡도 $O(2^n)$ 를 갖는 복잡한 문제
- NP-완전문제를 해결하려면 다음 중 한가지는 포기해야 한다.
  - 다항식 시간에 해를 찾는 것
  - 모든 입력에 대해 해를 찾는 것
  - 최적해를 찾는 것

근사 알고리즘

- 근사 알고리즘(Approximation algorithm)
  - NP-완전문제를 해결하기 위해 최적해를 포기하고 근사해를 찾는 방법
  - 근사해를 찾는 대신에 다항식 시간의 복잡도를 가진다.

- 근사 비율(Approximation ratio)
  - 근사 알고리즘의 근사해가 얼마나 최적해에 가까운지를 나타내는 근사 비율 (Approximation Ratio)을 알고리즘과 함께 제시하여야 한다.
  - 근사 비율은 근사해의 값과 최적해의 값의 비율로서, 1.0에 가까울수록 정확도가 높은 알고리즘이다.
  - 근사 비율을 계산하려면 최적해를 알아야 하는 모순이 생긴다.
    따라서 최적해를 대신할 수 있는 '간접적인' 최적해를 찾고,
    이를 최적해로 삼아 근사 비율을 계산한다.

TSP를 위한 근사 알고리즘

- TSP를 위한 근사 알고리즘을 고안하려면, 먼저 다항식 시간 알고리즘을 가지면서 유사한 특성을 가진 문제를 찾아야 한다.
  - TSP와 비슷한 특성을 가진 문제는 MST (Minimum Spanning Tree, 최소 신장 트리) 문제이다.
  - MST는 모든 점을 사이클 없이 연결하는 트리 중에서 트리 선분의 가중치 합이 최소인 트리이다.
  - MST의 모든 점을 연결하는 특성과 최소 가중치의 특성을 TSP에 응용하여, 시작 도시를 제외한 다른 모든 도시를 트리 선분을 따라 1번씩 방문하도록 경로를 찾는다.

여행자 문제(TSP)를 위한 근사 알고리즘

Approx_MST_TSP

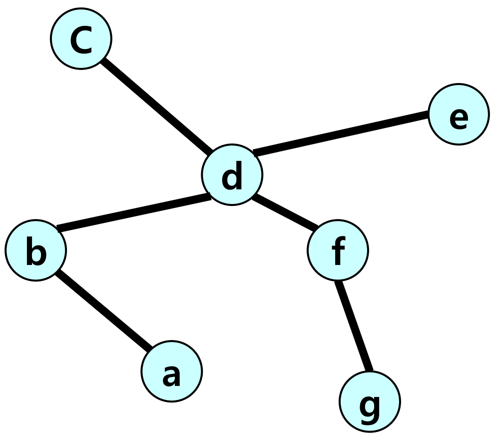
입력: n개의 도시, 각 도시간의 거리

출력: 출발 도시에서 각 도시를 1번씩만 방문하고
출발 도시로 돌아오는 도시 순서

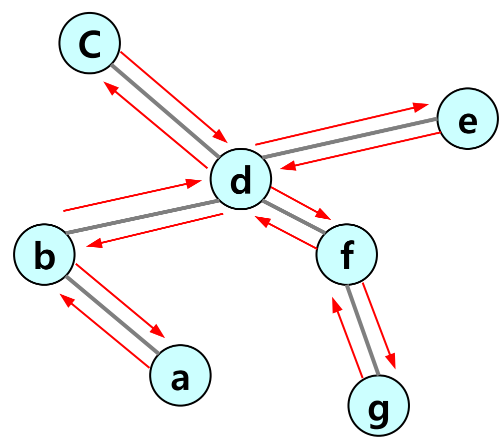
1. 입력에 대하여 MST를 찾는다.
2. MST에서 임의의 도시에서 출발하여
   트리의 선분을 따라 모든 도시를 방문하고
   출발했던 도시로 되돌아오는 도시 방문 순서를 찾는다.
3. return 이전 단계에서 찾은 도시 방문 순서에서
   중복되어 나타나는 도시를 제거한 도시 순서
   (단, 도시 순서 마지막의 출발 도시는 제거하지 않는다.)

Approx_MST_TSP로 근사해를 찾는 과정

1. 먼저 MST를 찾는다.

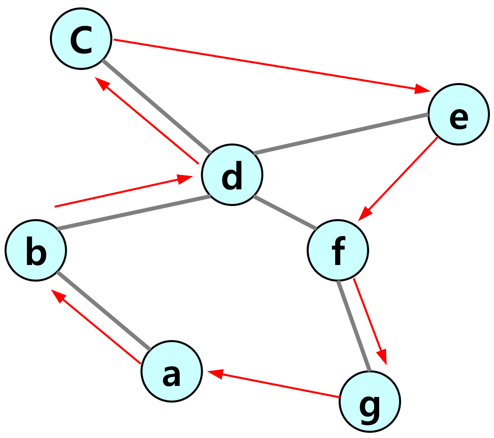


2. MST의 임의의 시작점(도시 1)에서 출발하여 트리의 선분을 따라서 모든 도시를 방문하고 돌아오는 방문 순서를 구한다.



[a b d c d e d f g f d b a]

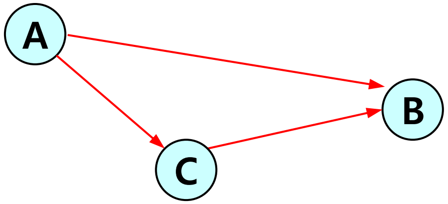
3. MST를 사용하여 구한 방문순서에서 가장 마지막에 있는 출발 도시 1을 제외하고, 중복 방문하는 도시를 모두 제거한다.



[a b d c ~~d~~ e ~~d~~ f g ~~f~~ ~~d~~ ~~b~~ a]

TSP 근사해 [a b d c e f g a]

- 중복 방문 도시를 제거하는 과정에 삼각 부등식 원리 적용
  - 삼각 부등식 원리:
    삼각형의 한 변의 길이는 다른 두 변의 길이의 합보다 짧다.
  - 즉, 도시 A에서 도시 B로 가는 거리는
    도시 A에서 다른 도시 C를 경유하여 도시 B로 가는 거리보다 짧다.



[A B]의 거리가 **[A C B]**의 거리보다 짧다.

시간복잡도

- Line 1: MST를 찾는 데에는 Kruskal 알고리즘이나 Prim 알고리즘의 시간복잡도만큼 시간이 걸린다.
- Line 2: 트리 선분을 따라서 도시 방문 순서를 찾는 데는 $O(n)$ 시간이 걸린다. 왜냐하면, 트리의 선분 수가 $(n-1)$이므로
- Line 3: line 2에서 찾은 도시 방문 순서를 따라가며, 단순히 중복된 도시를 제거하므로 $O(n)$ 시간이 걸린다.
- 따라서, 시간복잡도는 (Kruskal알고리즘이나 Prim알고리즘의 시간복잡도)+$O(n) + O(n)$이므로 Kruskal알고리즘이나 Prim알고리즘의 시간복잡도와 같다.
  - Kruskal 알고리즘의 시간복잡도: $O(\mathrm{m} \log m)$, m은 선분의 수
  - Prim 알고리즘의 시간복잡도: $O(n^2)$, n은 점의 수

근사 비율

- TSP의 최적해를 알 수 없으므로, MST 선분의 가중치의 합 M을 '간접적인' 최적해의 값으로 활용한다.
  - TSP의 최적해가 있다고 할 때, TSP에서 출발지로 되돌아오는 마지막 선분을 제거하면, 그것은 모든 정점을 연결하는 ST(Spanning Tree)가 된다.
  - MST의 가중치의 합 M은 모든 ST의 가중치의 합 중에 가장 작은 값이므로 TSP의 가중치의 합보다 항상 작다고 할 수 있다.
- Approx_MST_TSP로 계산한 근사해의 값은 2M보다 크지 않다.
  - line2에서 MST의 선분을 따라서 구한 도시 방문 순서는 모든 MST 선분이 2번 사용되므로 경로 길이는 2M이다.
  - line3에서 삼각 부등식 원리에 의해 중복 방문 도시를 제거하므로 최종 도시 방문 순서의 길이는 항상 2M보다 크지 않다.
- 따라서 이 알고리즘의 근사비율은 2M/M=2보다 크지 않다.
  즉, 근사해의 값이 최적해의 값의 2배를 넘지 않는다.

Q&A