그래프
(Graph)

- 최단경로(shortest path) 문제
- 최소 스패닝트리 (minimum spanning tree) 문제

8. 에지에 가중치가 있는 그래프에서의 최단경로 문제

- 에지에 가중치가 있는 그래프(weighted graph)에서 두 정점 u와 v에 대하여 u로부터 v까지의 최단경로를 구하는 문제

- **Dijkstra 알고리즘**
  - **에지의 가중치가 양수인 그래프**에서 source s로부터 다른 모든 정점까지의 최단경로를 찾는 알고리즘
  - s로부터의 최단경로 길이가 증가하는 순서대로 정점들을 고려하여 최단경로를 찾음
  - 욕심쟁이 알고리즘

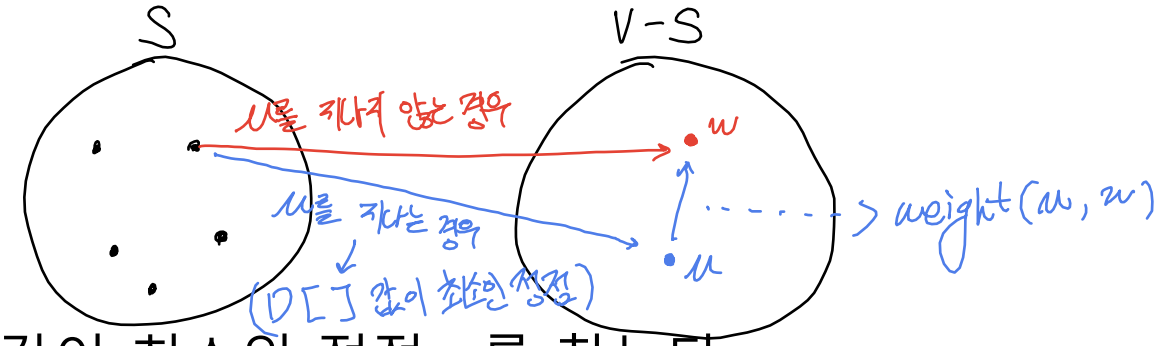
Dijkstra 알고리즘 (최단경로 찾는 알고리즘)

// 그래프는 연결되어 있다고 가정 (그렇지 않은 경우, 연결요소를 구하여 수행)
// V : 정점들의 집합
// n: 정점들의 수
// S : source s로부터 현재 최단경로를 찾은 정점들의 집합
// D[v]: s로부터 S에 속한 정점들만을 통하여 정점 v까지 가는 최단경로의 길이

S = {s}
while (|S| ≠n)
  V – S의 정점들 중 D[] 값이 최소인 정점 u를 찾는다.
  // 그러면 D[u]는 s로부터 u까지의 최단경로의 길이가 된다.
  S = S ∪ {u}
  u로부터 인접한 V – S의 각 정점 w에 대하여,
    D[w] = min {D[w], D[u] + weight(u, w)}

Dijkstra 알고리즘 (최단경로 찾는 알고리즘)

// 그래프는 연결되어 있다고 가정 (그렇지 않은 경우, 연결요소를 구하여 수행)
// V : 정점들의 집합
// n: 정점들의 수
// S : source s로부터 현재 최단경로를 찾은 정점들의 집합
// D[v]: s로부터 S에 속한 정점들만을 통하여 정점 v까지 가는 최단경로의 길이



S = {s}
while (|S| ≠n)
  V – S의 정점들 중 D[] 값이 최소인 정점 u를 찾는다.
  // 그러면 D[u]는 s로부터 u까지의 최단경로의 길이가 된다.
  S = S ∪ {u}
  u로부터 인접한 V – S의 각 정점 w에 대하여,
      if (D[u] + weight(u, w) < D[w])
          D[w] = D[u] + weight(u, w)
          parent[w] = u

Dijskstra 알고리즘의 정확성

정리: 에지 가중치가 양수인 연결된 그래프에서 Dijkstra 알고리즘은 모든 정점에 대하여 최단 경로를 구한다.

증명: 정점 s로부터 정점 u까지 최단경로의 길이를 δ(s, u)로 표기한다. S에 있는 모든 정점 v에 대하여, D[v]가 δ(s, v)와 같음을 보인다. 이는 S에 포함되는 정점 개수에 대한 수학적 귀납법으로 증명한다.

Dijskstra알고리즘의 정확성

S에 포함되는 정점 개수가 1일 경우, 자명하다.

S에 포함되는 정점 개수가 k-1일 때, S의 모든 정점 v에 대하여 D[v] = δ(s, v)라 하자. S에 포함되는 정점들의 수가 k일 때 보인다.

V-S 정점들 중 D[]값이 최소인 정점을 u라 하자.

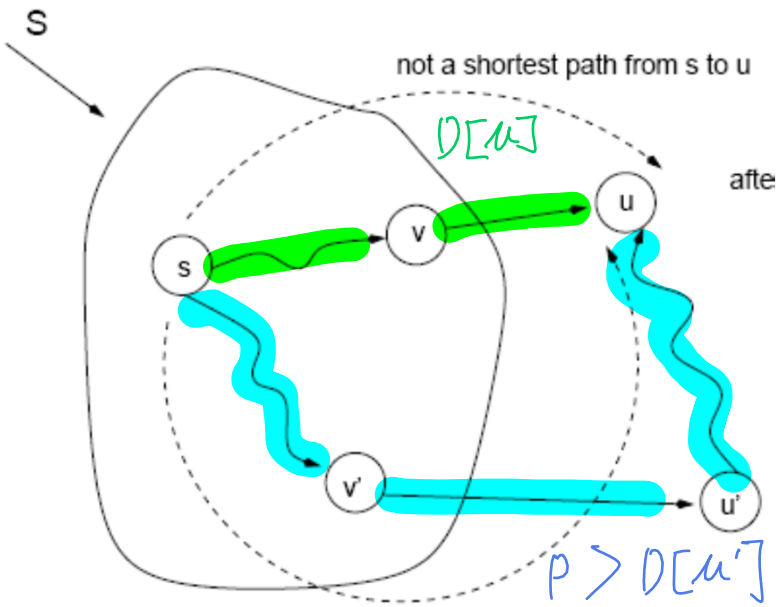
D[u] = δ(s, u), 즉 D[u]가 s로부터 u까지 가는 최단경로 길이와 같음을 보인다.

만약 D[u] ≠ δ(s, u)라 가정하자. 그렇지 않다고 가정하고, 모순됨을 보인다.

Dijskstra알고리즘의 정확성

s로부터 u까지 최단경로를 P라 하면, 가정에 의하여 P의 길이는 D[u]보다 작다. P는 V-S 에 있는 u가 아닌 어떤 정점을 지난다.

경로 P에서, S에 포함되지 않는 정점 중 처음으로 지나는 정점을 u' 라 하자.



D[u'] < P의 길이 < D[u]이다.

그런데 Dijkstra 알고리즘에서, V-S에 있는 모든 정점들 중 D[u]가 최소이다.

이는 모순이다. 따라서 S에 포함되는 정점들의 수가 k일 때도, S의 모든 정점 v에 대하여 D[v] = δ(s, v)이다.

Dijkstra 알고리즘 구현 – 인접행렬 표현

```
// The weighted graph is represented as an adjacency matrix A
// 최단경로를 찾기 위하여 배열 parent를 이용
// S: 최단경로를 찾은 vertex들의 집합

S = {s}
for i = 0 to n
    D[i] = adjMatrix[s][i]   // 지나는 edge 없으면 ∞로 초기화
    parent[i] = s            // S로 우선 초기화

while (|S| ≠n)
    V – S 의 vertex 들 중 D[] 값이 최소인 정점을 u라 하자
    S = S ∪{u}
    u로부터 인접한 모든 정점 w에 대하여
        if(D[w] > (D[u] + adjMatrix[u][w])
            D[w] = D[u] + adjMatrix[u][w]
            parent[w] = u
```

parent 값 update

O(log₂n)

n² log n

- 수행시간 분석:

n: 정점 수

m: 간선(에지) 수

➢ 그래프를 인접행렬로 표현할 경우

수행시간: $O(n^2)$

➢ 그래프를 인접리스트로 표현할 경우

(1) D[]를 단순 배열(리스트)로 관리할 경우

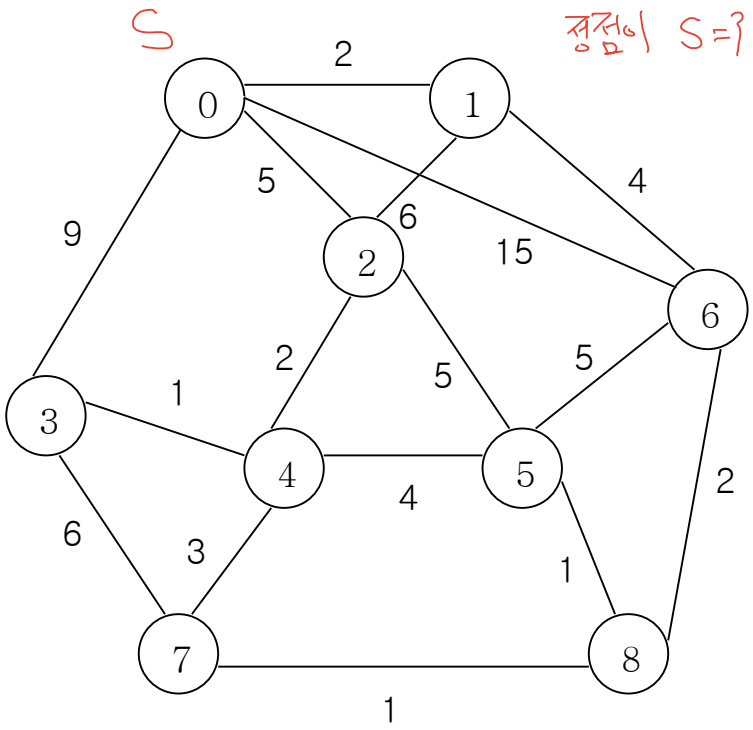
수행시간 $O(n^2)$

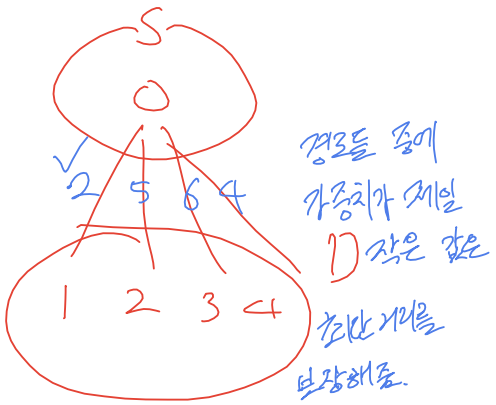
(2) D[]를 최소힙으로 관리할 경우

수행시간: $O(m \log n)$

Dijkstra 알고리즘 예

예:





Source: 0

배열 D

0	1	2	3	4	5	6	7	8
0	2	5	9	∞	∞	15	∞	∞

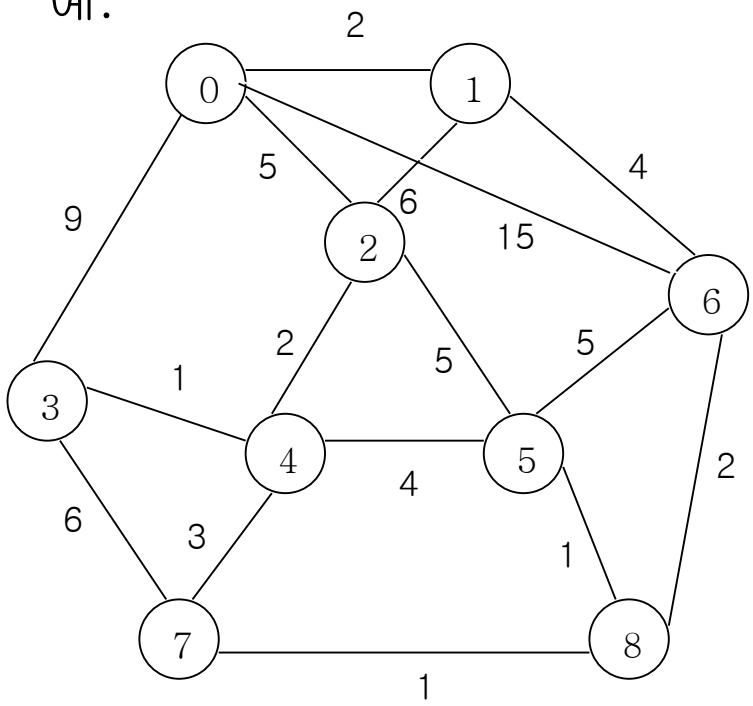
S = { 0 }

배열 parent

0	1	2	3	4	5	6	7	8
0	0	0	0	-	-	0	-	-

예

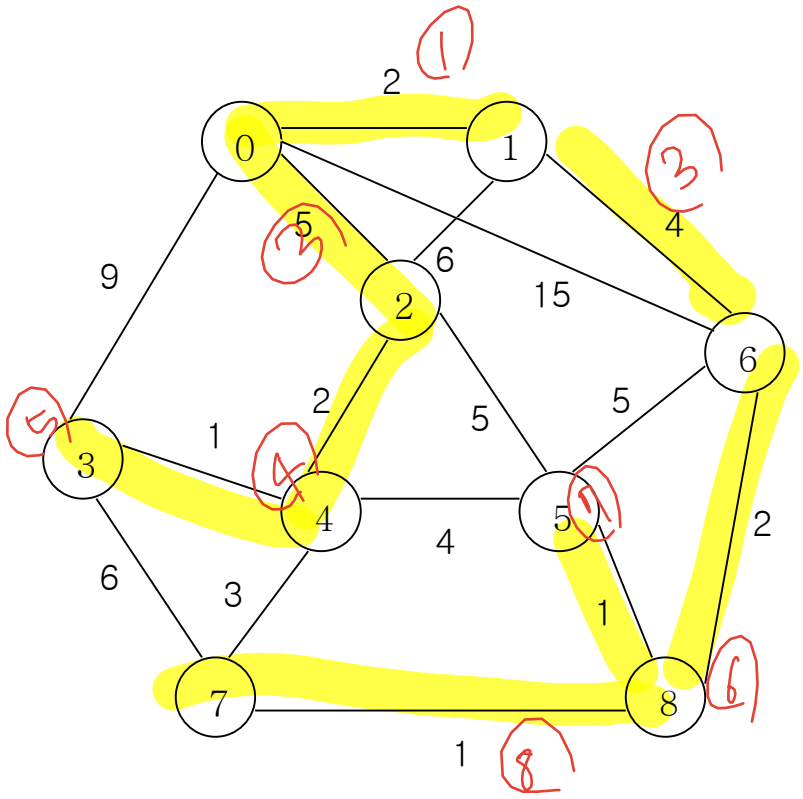
예:



Source: 0

0	1	2	3	4	5	6	7	8	
0	2	5	9	∞	∞	15	∞	∞	S = { 0 }
0	2	5	9	∞	∞	6	∞	∞	S = { 0, 1 }
0	2	5	9	7	10	6	∞	∞	S = { 0, 1 , 2}
0	2	5	9	7	10	6	∞	8	S = { 0, 1 , 2, 6}

예



Source: 0

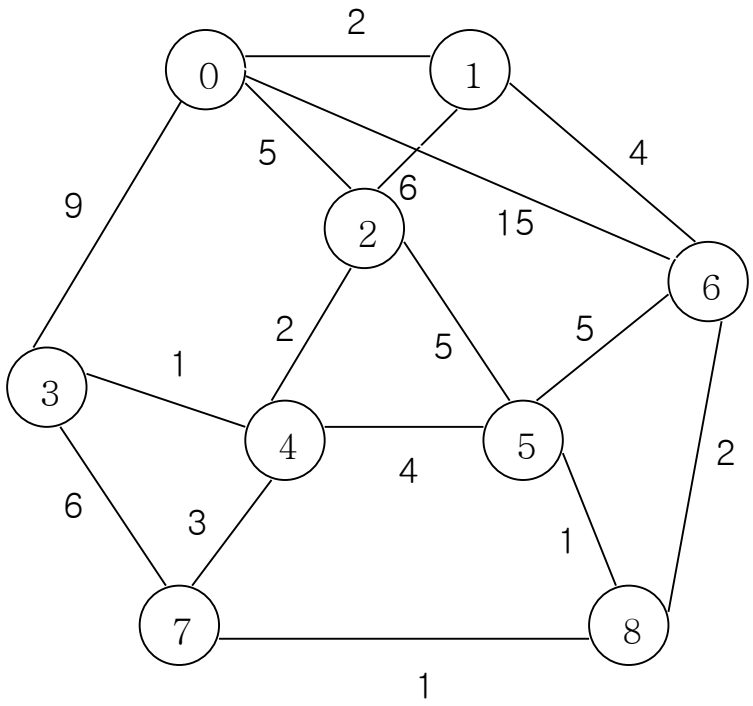
0	1	2	3	4	5	6	7	8	
0	2	5	9	7	10	6	∞	8	S = { 0, 1 , 2, 6}

0	1	2	3	4	5	6	7	8	
0	2	5	8	7	10	6	10	8	S = { 0, 1 , 2, 6, 4}

0	1	2	3	4	5	6	7	8	
0	2	5	8	7	10	6	10	8	S = { 0, 1 , 2, 6, 4, 3}

0	1	2	3	4	5	6	7	8	
0	2	5	8	7	9	6	9	8	S = { 0, 1 , 2, 6, 4, 3, 8}

예



Source: 0

S = { 0, 1 , 2, 6, 4, 3, 8}

0	1	2	3	4	5	6	7	8
0	2	5	8	7	9	6	9	8

S = { 0, 1 , 2, 6, 4, 3, 8, 5}

0	1	2	3	4	5	6	7	8
0	2	5	8	7	9	6	9	8

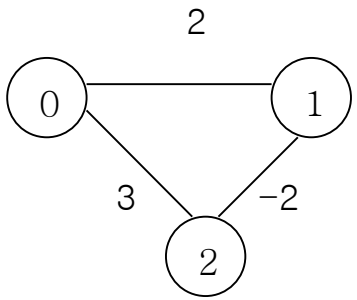
S = { 0, 1 , 2, 6, 4, 3, 8, 5, 7}

0	1	2	3	4	5	6	7	8
0	2	5	8	7	9	6	9	8

Note: 만약 가중치가 음인 에지가 있을 경우 Dijkstra 알고리즘은 최단경로를 찾지 못할 수도 있다.

다음 예에서, source 0으로부터 1까지 가는 최단경로를 구할 때, Dijkstra 알고리즘은 최단경로로 길이 2인 경로 (0,1)을 구한다.

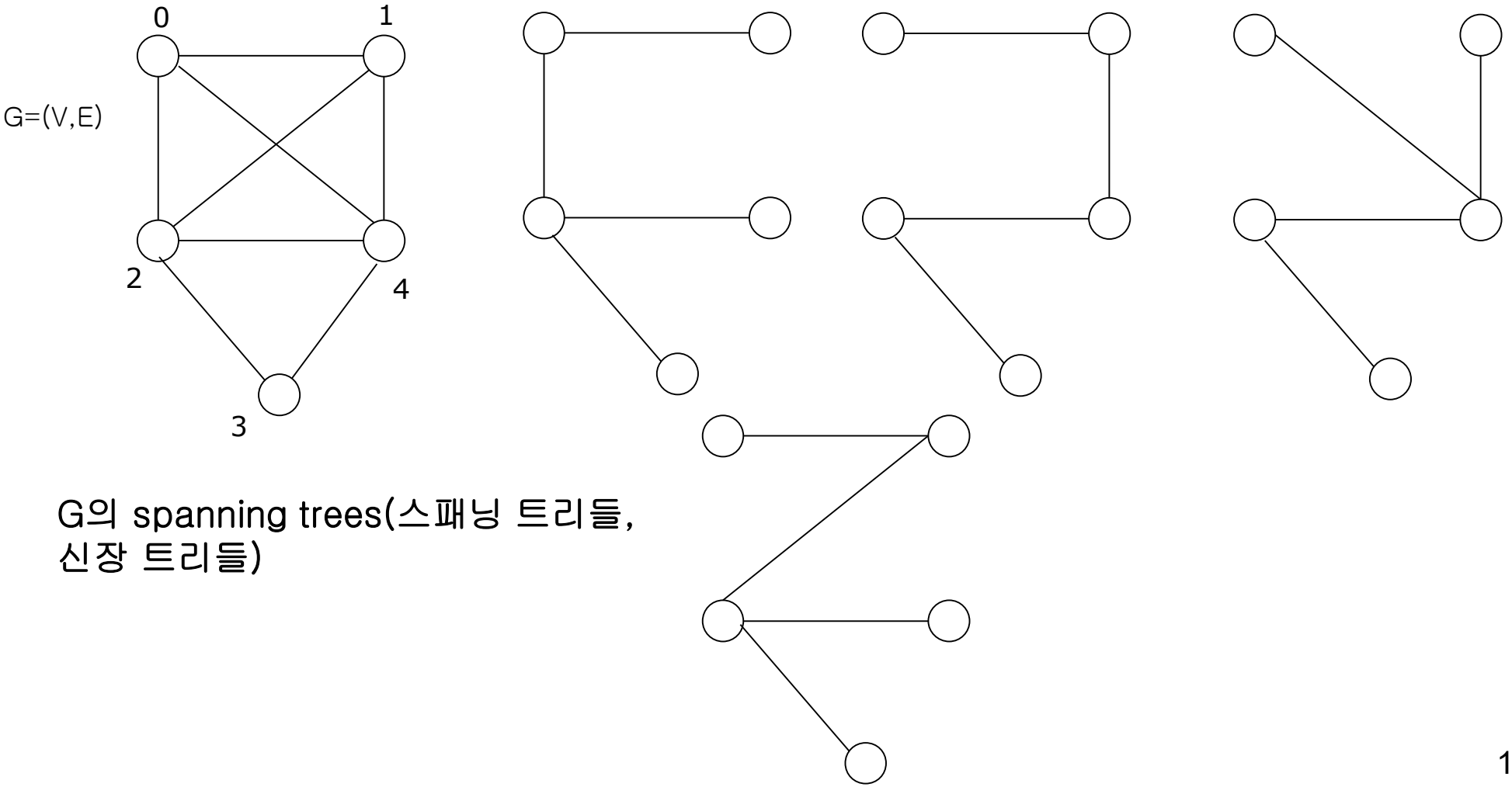
실제 최단경로는 길이 1인 (0, 2, 1)이다.



8. 최소비용 스패닝(신장) 트리

트리는 사이클이 없는 연결된 그래프이다.

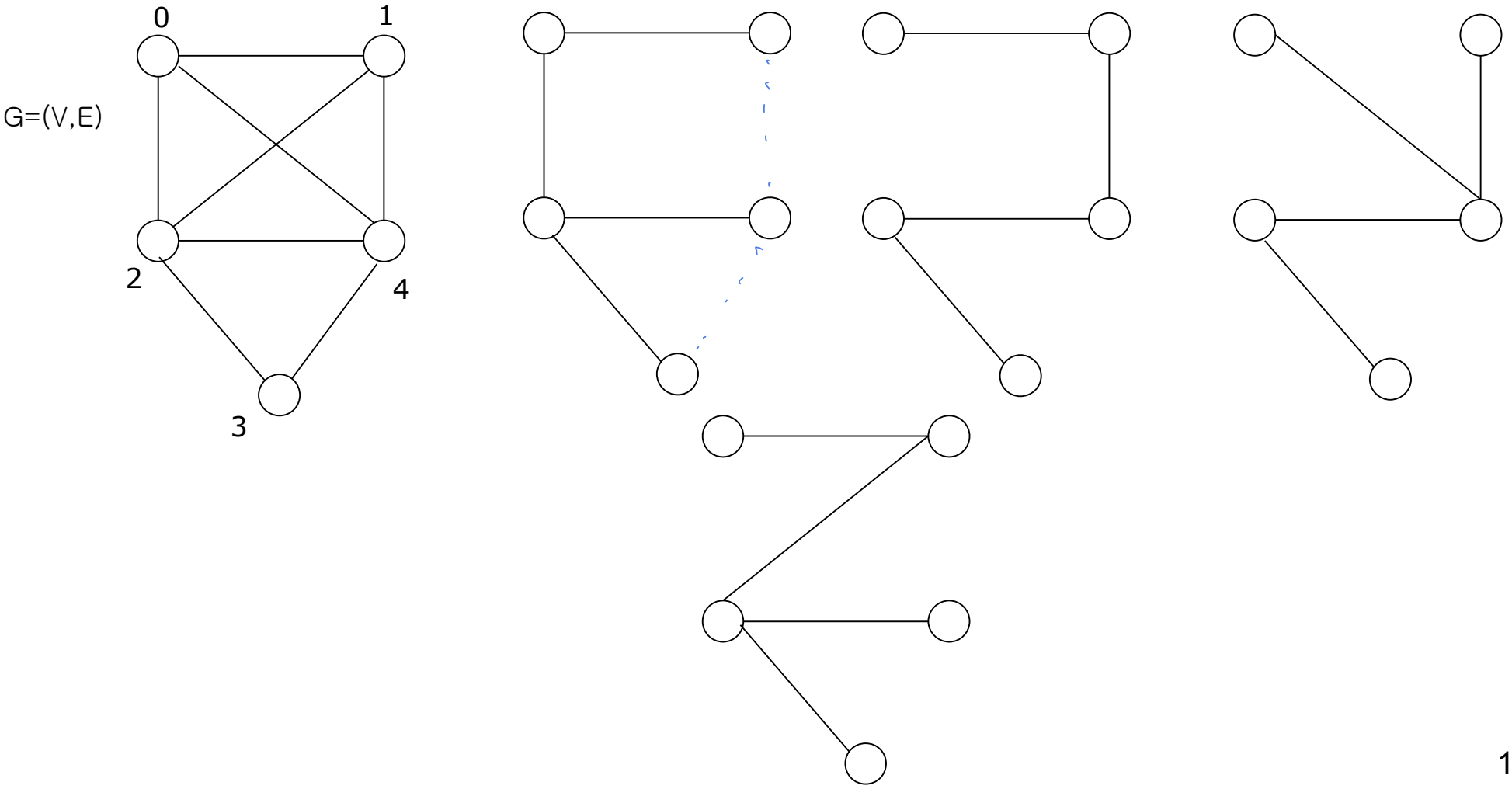
정의: 그래프 G의 스패닝 트리(spanning tree 혹은 신장 트리)는 G의 부그래프로서 G의 모든 정점을 포함하는 트리이다.



G의 spanning trees(스패닝 트리들, 신장 트리들)

최소비용 스패닝(신장) 트리

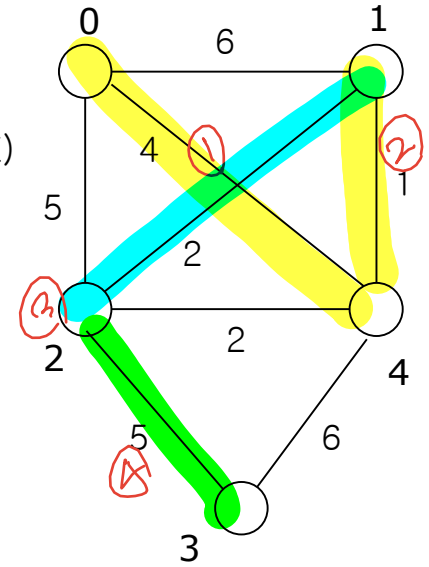
그래프의 에지 하나를 스패닝 트리에 추가하면 사이클이 생긴다.



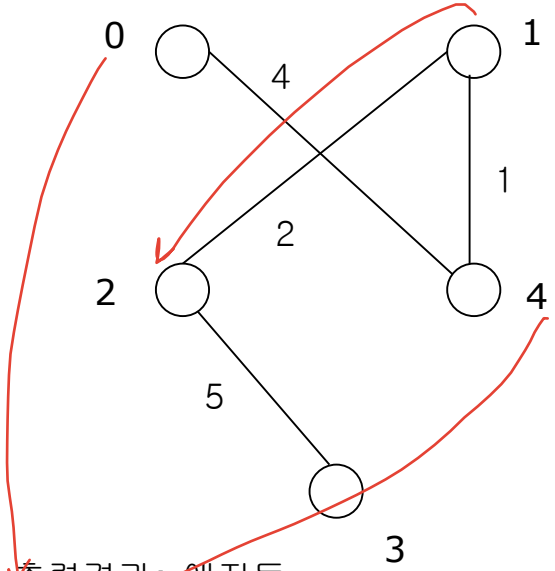
최소비용 스패닝(신장) 트리

- 정의: 에지에 가중치가 있는 그래프 G의 최소비용 스패닝트리(minimum spanning tree)는 전체 가중치(가중치 합)가 최소인 G의 스패닝트리이다.

G=(V,E)



G의 minimum spanning tree



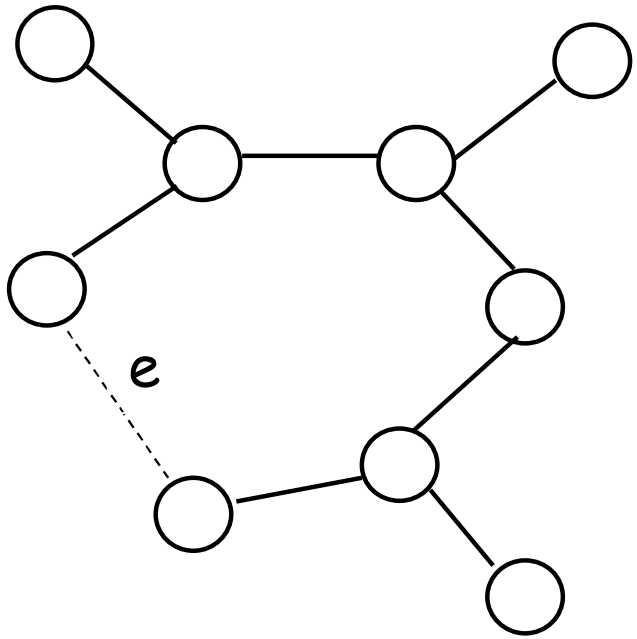
출력결과: 에지들

0 4
1 2
1 4
...

에지수 = 정점수 - 1

스패닝 트리의 성질

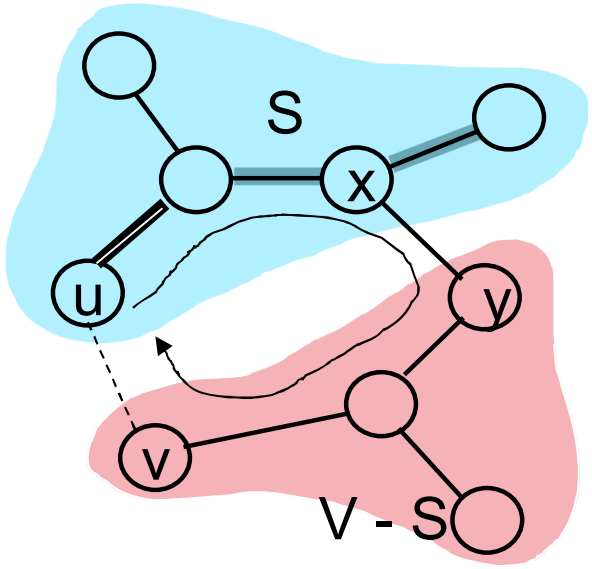
보조정리: 트리 T에 대하여, T에 있지않는 에지 e를 T에 추가하면 사이클이 생긴다.



최소 비용 스패닝 트리의 성질

보조정리: 연결된 가중치 그래프 G = (V, E, W)에 대하여, S를 V의 임의의 부분집합이라 하자. S와 V-S사이의 에지들 중 가중치가 가장 작은 에지 e = (u,v)를 포함하는 최소비용 신장트리(MST)가 존재한다.

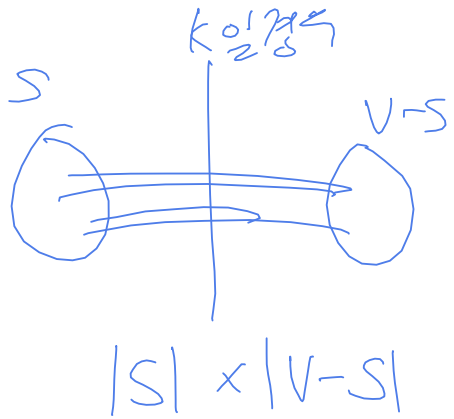
증명: T를 임의의 최소비용 스패닝 트리(MST)라 하자. T가 e를 포함하지 않는다고 하자. T에 e를 추가하면 사이클 C가 만들어진다. 이 사이클 C는 S와 V-S사이의 어떤 에지 e′ = (x,y)를 포함한다.



T′ = (T – {e′}) U {e}는 스패닝 트리이다.
W(T): T의 에지 가중치 합
W(T′): T′의 에지 가중치 합
weight(e): 에지 e의 가중치
weight(e′): 에지 e′의 가중치
weight(e) ≤ weight(e′)이므로, W(T′) ≤ W(T)이다.
따라서 에지 e′를 포함하는 최소비용 신장트리가 존재한다.

Prim의 알고리즘 (욕심쟁이 알고리즘)

// G = (V, E): 에지에 가중치가 있는 연결된 그래프
// n: |V|, m: |E|
// T: 최소비용신장트리



임의의 정점 s를 선택한다.
S = {s} // 초기의 트리 T의 정점들의 집합
A = ∅ // 초기의 트리 T의 에지들의 집합
parent[s] = -1

while (A의 에지 수 ≠ n-1)

S와 V-S사이의 최소 가중치의 에지 e = (u,v)를 선택한다.

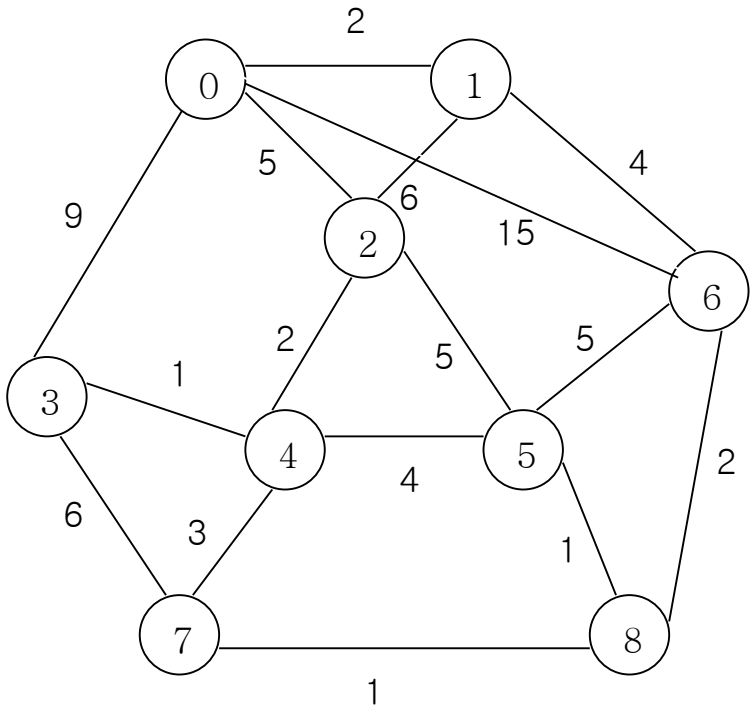
최대 O(k) ~ O(n²)

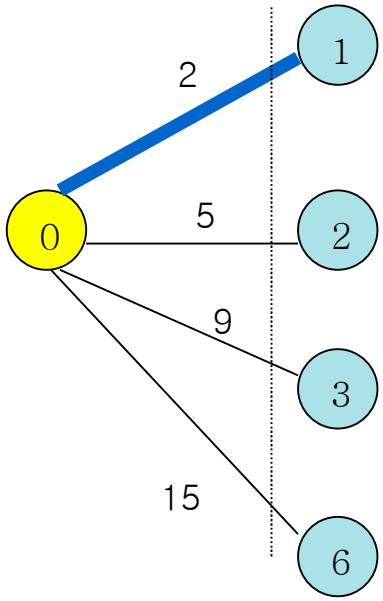
S = S U {v}

parent [v] = u
A = A U {e} // 트리 T에 e를 추가한다.

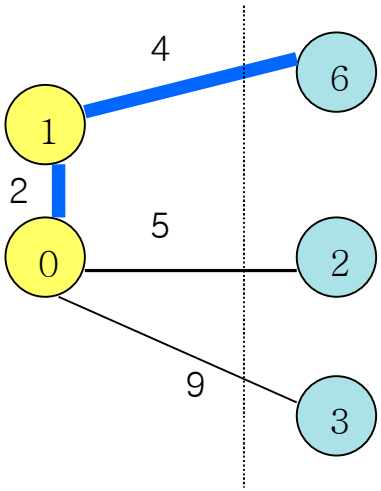
Prim 알고리즘 예 (1)

- 예





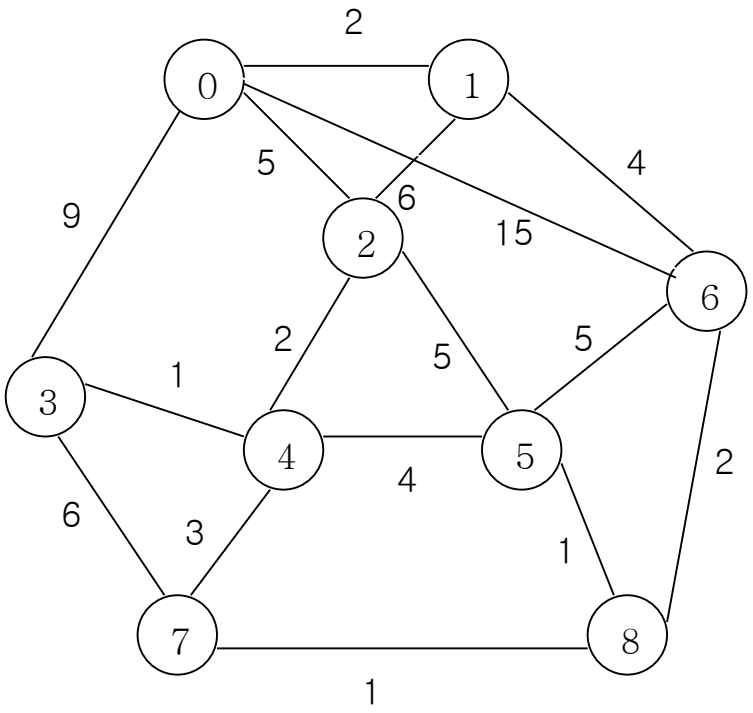
S = {0}
A = ∅

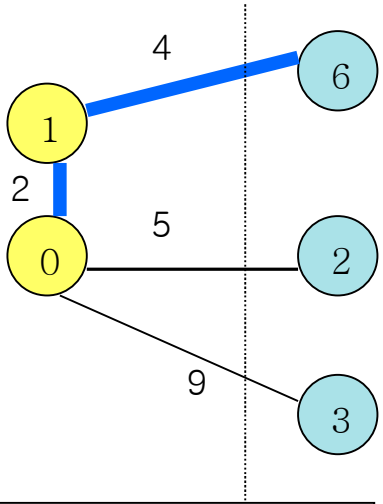


S = {0,1}
A = {(0,1)}

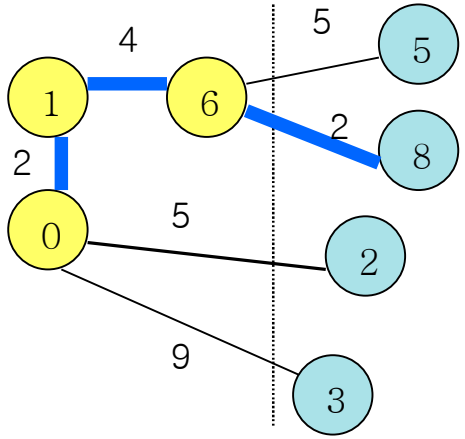
Prim 알고리즘 예 (2)

- 예

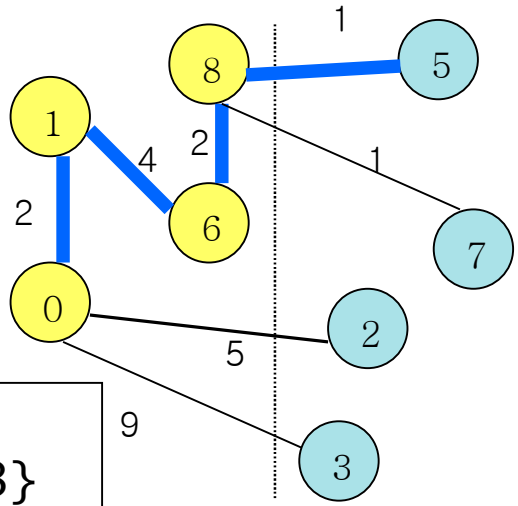




S = {0,1}
A = {(0,1)}



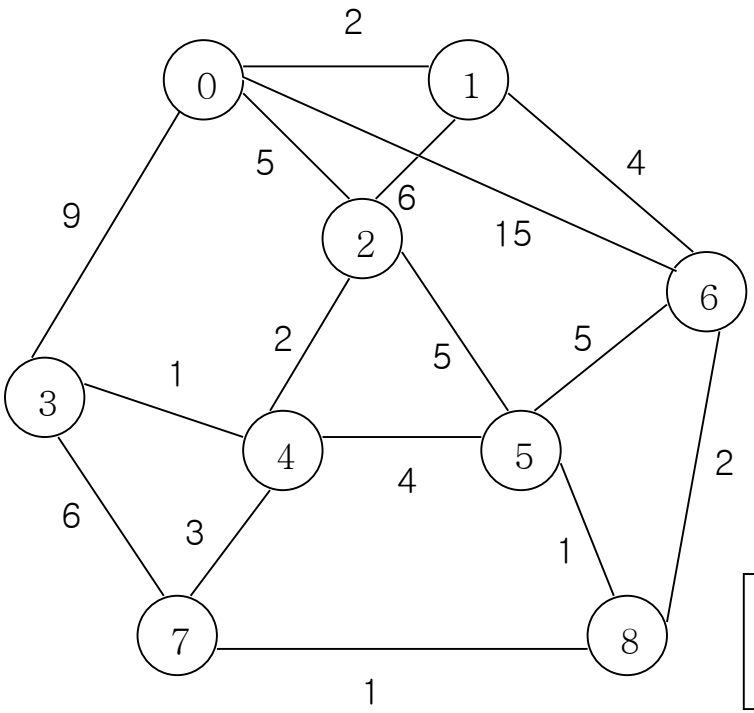
S = {0,1,6}
A = {(0,1),(1,6)}



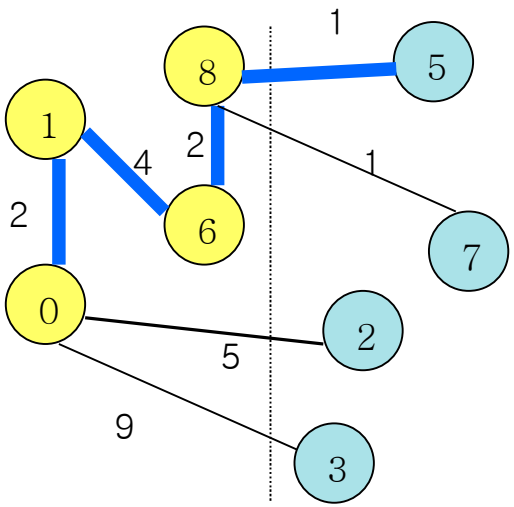
S = {0,1,6,8}
A = {(0,1),(1,6),(6,8}

Prim 알고리즘 예 (3)

- 예

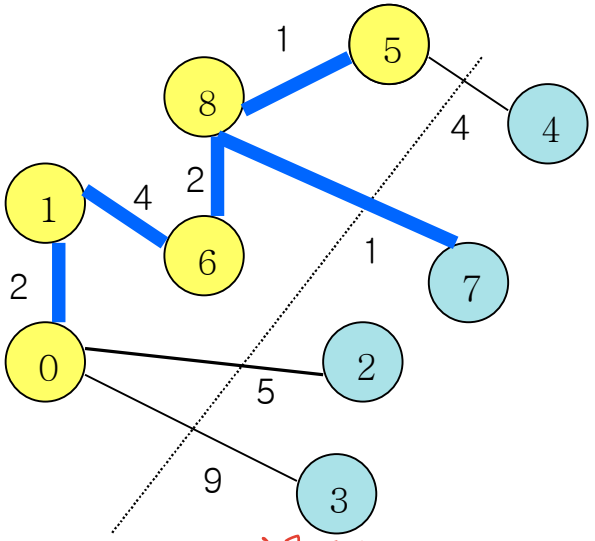


8→5
8→7 가는 비용이 같음.



8→5 경우

S = {0,1,6,8}
A = {(0,1),(1,6),(6,8}

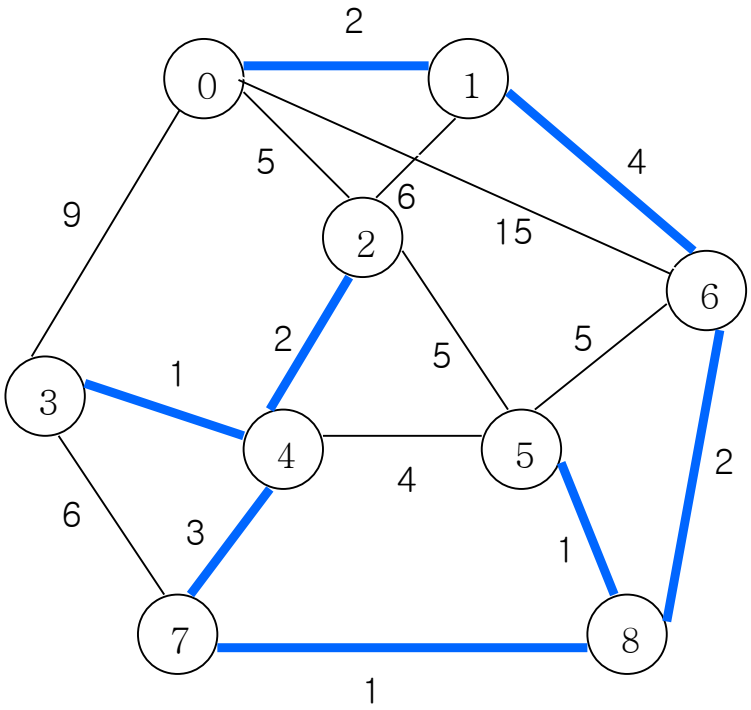


8→7 경우

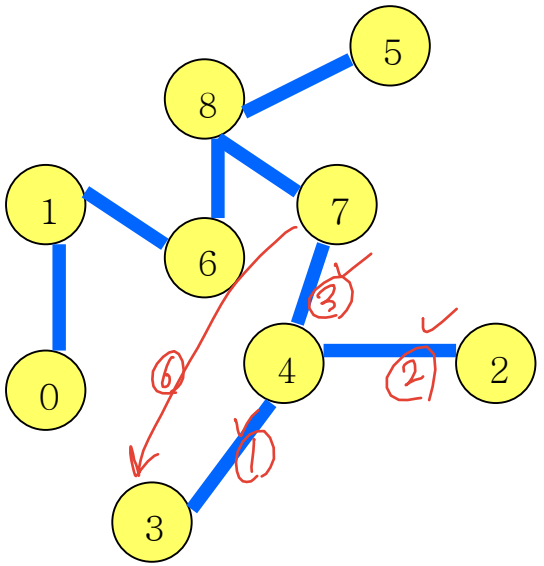
S = {0,1,6,8,5}
A = {(0,1),(1,6),(6,8),
(8,5)}

Prim 알고리즘 예 (4)

- 예



Prim 알고리즘에 의하여
구해진 스패닝 트리



S = {0,1,6,8,5, 7,4, 3,2}
A = {(0,1),(1,6),(6,8),(8,5),(8,7),(7,4),
(4,3),(4,2)}

Prim의 알고리즘 구현 1

```
//V-S에 속하는 정점 v에 대하여
//     minWeight[v]: v와 S의 정점들 사이의 에지들의 최소 가중치

V의 각 정점 v에 대하여,
    minWeight[u] = ∞
    parent[u] = -1

임의의 정점 s를 선택한다.
S = {s} // 트리 T의 정점들의 집합
A = ∅  // 트리 T의 에지들의 집합
s와 인접한 각 정점 v에 대하여,
    minWeight[v] = 에지 (s,v)의 가중치
    parent[v] = s
while (A의 에지 수 ≠ n-1)
    S와 V-S사이의 최소 가중치의 에지 e = (u,v)를 선택한다.
        => V-S의 정점들의 minWeight[]가 최소인 정점을 선택
    S = S U {v}
    Parent [v] = u
    A = A U {e} // T에 e를 추가한다.
    // V-S의 정점들의 minWeight[]를 upadate
    v와 인접한 V-S의 정점 w에 대하여,
        if (weight(v,w) < minWeight[w])
            minWeight[w] = weight(v,w)
            parent[w] = v
```

수행시간: $O(n^2)$

Prim의 알고리즘 구현 2 (최소힙 이용)

```
// V-S에 속하는 정점 v에 대하여
//      minWeight[v]: v와 S의 정점들 사이의 에지들의 최소 가중치
// V-S의 모든 정점들의 minWeight[]값을 최소힙 H으로 관리

V의 각 정점 v에 대하여,
    minWeight[u] = ∞
    parent[u] = -1

임의의 정점 s를 선택한다.
S = {s} // 트리 T의 정점들의 집합
A = ∅  // 트리 T의 에지들의 집합
s와 인접한 각 정점 v에 대하여,
    minWeight[v] = 에지 (s,v)의 가중치
    parent[v] = s

V-S의 모든 정점들과 그 minWeight값을 힙 H에 삽입
While (H가 empty가 아니면)
    u =  deleteMin(H)      // 최소힙 H에서 minWeight가 최소인 정점 삭제
    S = S U {u}
    Parent [v] = u
    A = A U {e} // T에 e를 추가한다.
    u와 인접한 V-S의 모든 정점 v에 대하여,
        if (weight(u,v) < minWeight[v])
            minWeight[v] = weight(u,v)
            최소힙 H를 update
            parent[v] = u
```

수행시간: O(m log n)

Kruskal 알고리즘 (욕심쟁이 알고리즘)

최소 가중치 에지들을 정렬함 → 계속 추가해 나감.
n-1 만족하면

```
// 입력: 에지에 가중치가 있는 연결된 그래프 G = (V, E)
R = E;      // R은 남아 있는 에지들의 집합
F = ϕ;      // 현재까지 구성한 포리스트 에지들의 집합
while (|F| != n-1)
    R에서 최소 가중치의 에지 (v,w)를 제거한다.
    if ((v,w)를 F에 추가할 때 사이클을 만들지 않으면)
        (v,w)를  F에 추가한다;
```

Kruskal 알고리즘 수행시간:
O(m log m)

Kruskal 알고리즘 예

- 예