ALOHA

알고리즘반 8주차 – Disjoint Set & MCST

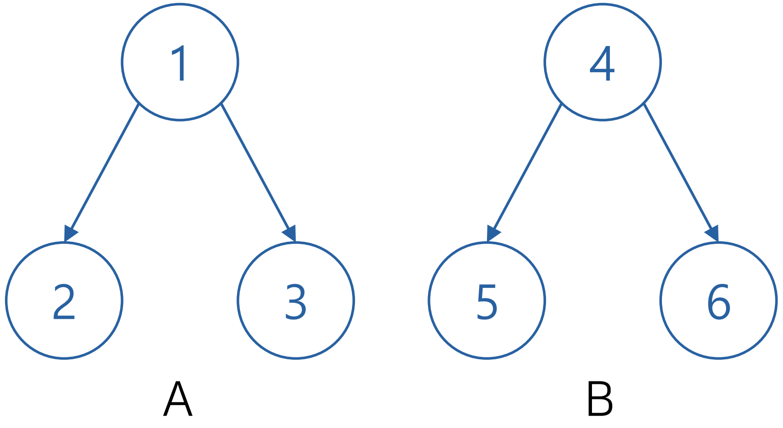
T. 성창호

Asst. 박주언 서유림

Disjoint Set

Disjoint Set

- 서로소 집합
  - 두 개 이상의 집합을 형성할 때, 교집합이 공집합이 되도록 구성하는 자료구조
  - Ex) A = {1,2,3}과 B={4,5,6}은 서로소 집합
- Disjoint Set은 대개 Tree로 구성한다
  - 항상 Parent-Child 관계와 Root Node가 존재한다.

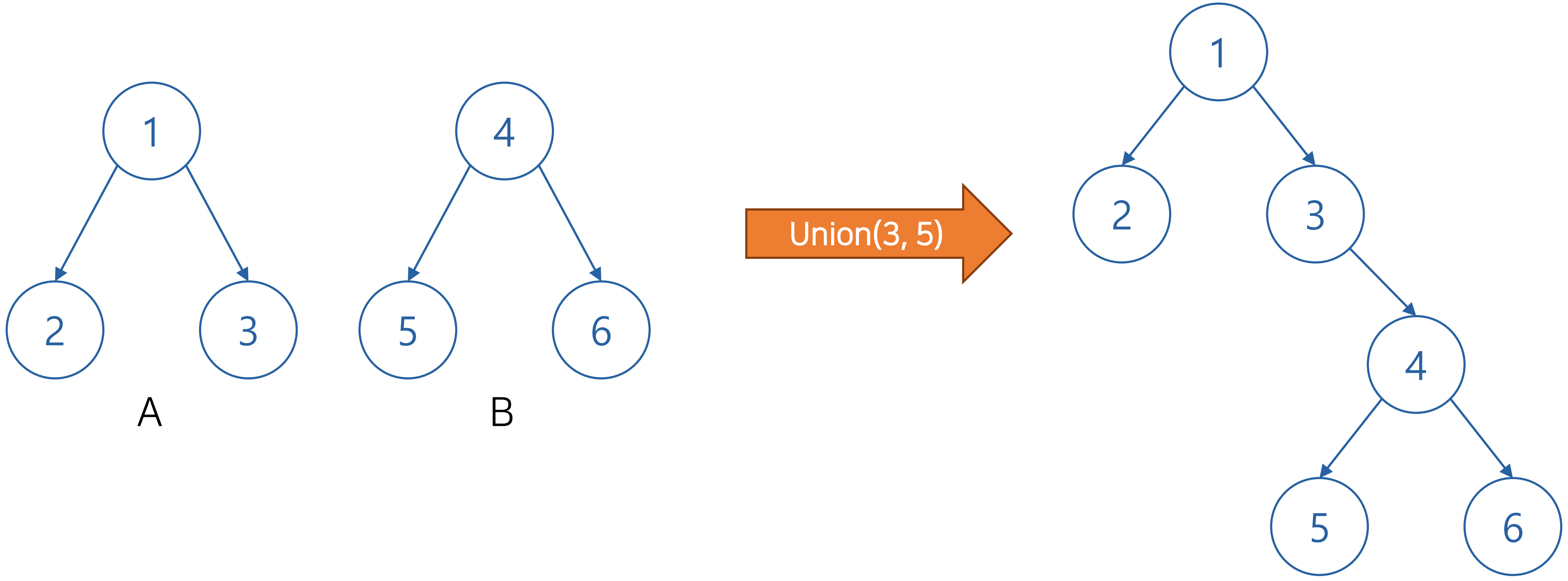


Disjoint Set – Union-Find

- Disjoint Set은 두 가지 연산을 지원한다
  1. Union(x, y)
     - x가 속하고 있는 집합과 y가 속하고 있는 집합을 합친다.
     - 이때 x와 y가 같은 집합 내에 있다면 합치지 않는다.
  2. Find(x)
     - x가 속하고 있는 집합을 구한다.

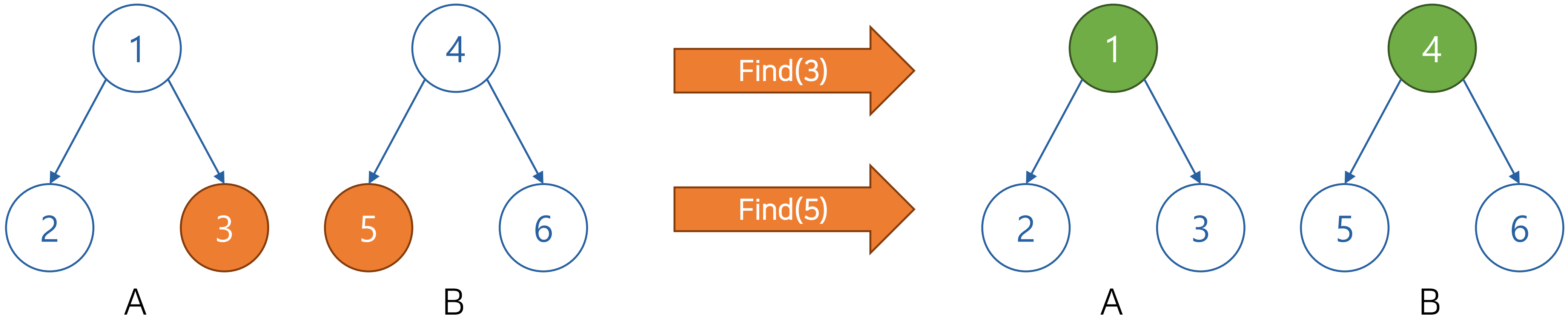
Disjoint Set – Union-Find

- Union(x, y)
  - x가 속하고 있는 집합과 y가 속하고 있는 집합을 합친다.



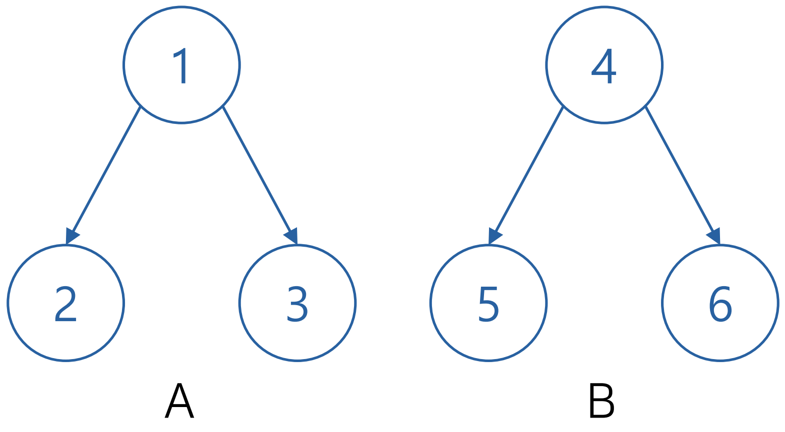
Disjoint Set – Union-Find

- Find(x)
  - X가 속하고 있는 집합을 구한다.
  - Root Node의 값을 집합의 대푯값으로 생각한다.
    - 집합을 구한다 == 해당 집합의 Root Node 값을 구한다



Disjoint Set – Union-Find

- Disjoint Set의 표현
  - Tree로 표현
    - 각각의 Node의 Parent Node를 저장한다.
  - 이때 배열이나 vector를 활용한다.



Node	1	2	3	4	5	6
Parent	NIL	1	1	NIL	4	4

Disjoint Set – Union-Find

- Union-Find
  - Union
    - 두 원소의 Root Node를 먼저 확인한 후, 두 원소의 Root Node가 다르다면, 두 원소가 속하고 있는 집합이 서로 다른 집합이고 Union(합침)이 가능하다.
  - Find
    - 원소의 Parent Node를 따라 올라가면서 더 이상 올라갈 수 없을 때, 즉 Root Node에 도달했을 때 값을 return한다.

Disjoint Set – Union-Find

Node	1	2	3	4	5	6	7	8	9
Parent	1	2	3	4	5	6	7	8	9

1. 초기의 모든 Node들의 Parent Node를 자기 자신으로 설정한다

2. 이때 자기 자신을 Parent Node로 삼는 Node, 즉 자기 자신을 가리키는 Node는 Root Node로 명명한다.

Disjoint Set – Union-Find

Node	1	2	3	4	5	6	7	8	9
Parent	1	2	3	4	5	6	7	8	9

1. 초기의 모든 Node들의 Parent Node를 자기 자신으로 설정한다
2. 이때 자기 자신을 Parent Node로 삼는 Node, 즉 자기 자신을 가리키는 Node는 Root Node로 명명한다.

Disjoint Set – Union-Find

Node	1	2	3	4	5	6	7	8	9
Parent	1	2	3	4	5	6	7	8	9

- Union(1, 3)

Disjoint Set – Union-Find

Node	1	2	3	4	5	6	7	8	9
Parent	1	2	3	4	5	6	7	8	9

- Union(1, 3)
  1. 1과 3의 Root Node를 구한다.
     Find(1), Find(3)

Disjoint Set – Union-Find

Node	1	2	3	4	5	6	7	8	9
Parent	1	2	3	4	5	6	7	8	9

- Union(1, 3)
  1. 1과 3의 Root Node를 구한다.
     Find(1), Find(3)
  2. 두 개의 Root Node를 서로 합친다.

Disjoint Set – Union-Find

Node	1	2	3	4	5	6	7	8	9
Parent	1	2	1	4	5	6	7	8	9

- Union(1, 3)
  1. 1과 3의 Root Node를 구한다.
     Find(1), Find(3)
  2. 두 개의 Root Node를 서로 합친다.
     (이때, 하나의 Root Node의 Parent Node를 다른 Root Node로 설정한다.)

Disjoint Set – Union-Find

Node	1	2	3	4	5	6	7	8	9
Parent	1	2	1	4	5	6	7	8	9

- Union(1, 5)
  1. 1과 5의 Root Node를 구한다.
     Find(1), Find(5)

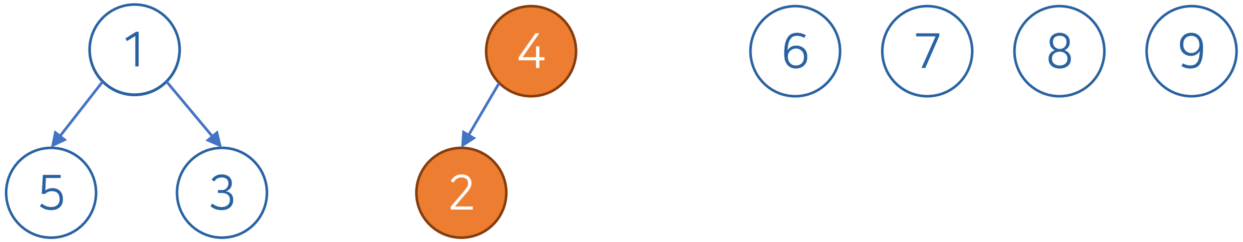
Disjoint Set – Union-Find

Node	1	2	3	4	5	6	7	8	9
Parent	1	2	1	4	1	6	7	8	9

- Union(1, 5)
  1. 1과 5의 Root Node를 구한다.
     Find(1), Find(5)
  2. 두 개의 Root Node를 서로 합친다.
     (이때, 하나의 Root Node의 Parent Node를 다른 Root Node로 설정한다.)

Disjoint Set – Union-Find

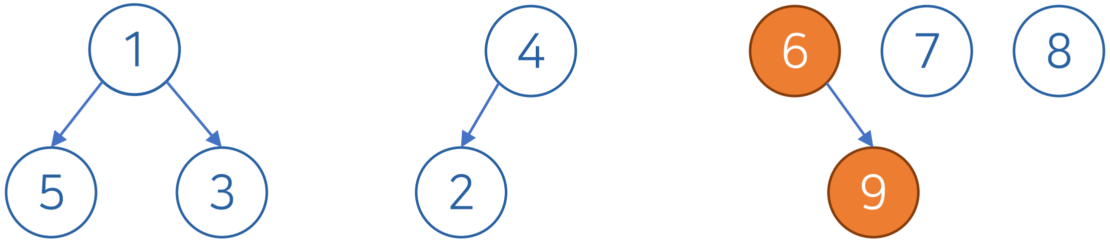
Node	1	2	3	4	5	6	7	8	9
Parent	1	4	1	4	1	6	7	8	9



- Union(4, 2)

Disjoint Set – Union-Find

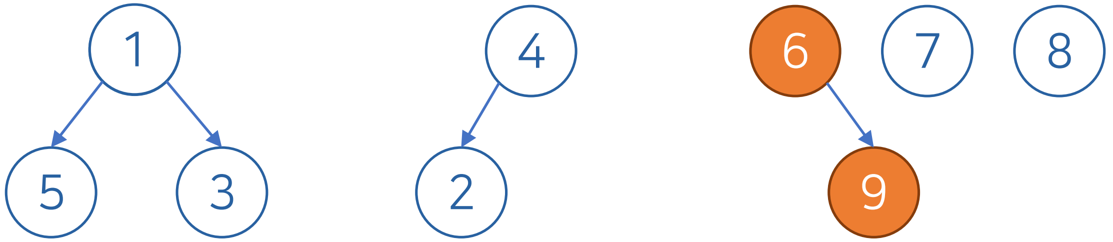
Node	1	2	3	4	5	6	7	8	9
Parent	1	4	1	4	1	6	7	8	6



- Union(6, 9)

Disjoint Set – Union-Find

Node	1	2	3	4	5	6	7	8	9
Parent	1	4	1	4	1	6	7	8	6

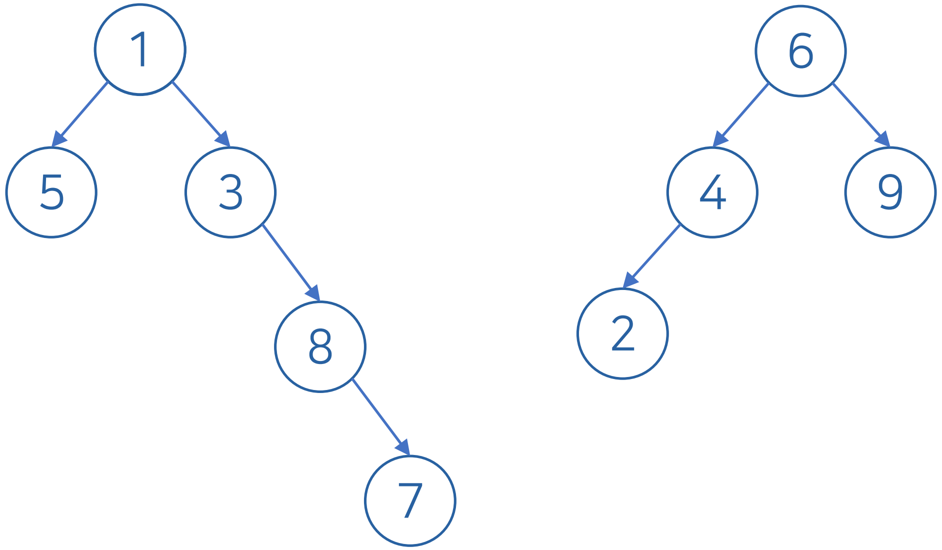


- Union(6, 9)

여러 번 반복…

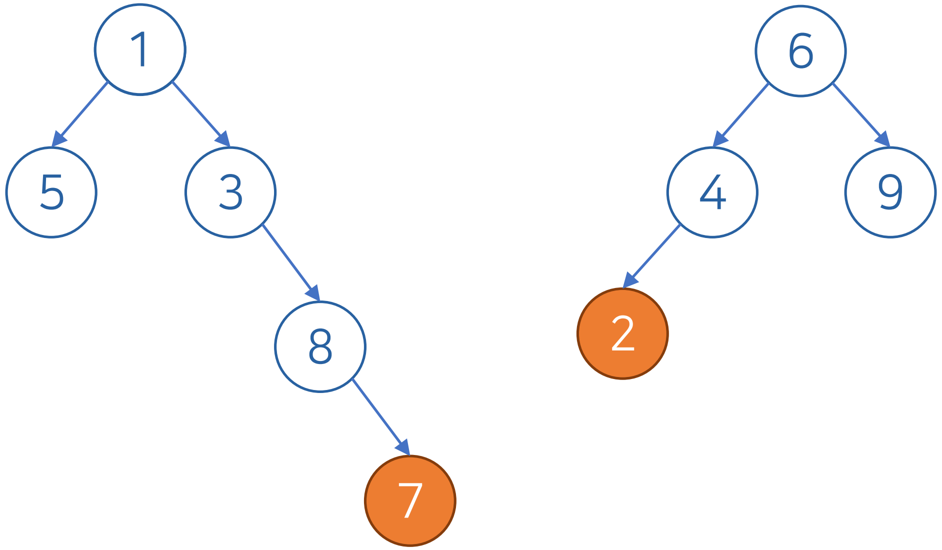
Disjoint Set – Union-Find

Node	1	2	3	4	5	6	7	8	9
Parent	1	4	1	6	1	6	7	3	6



Disjoint Set – Union-Find

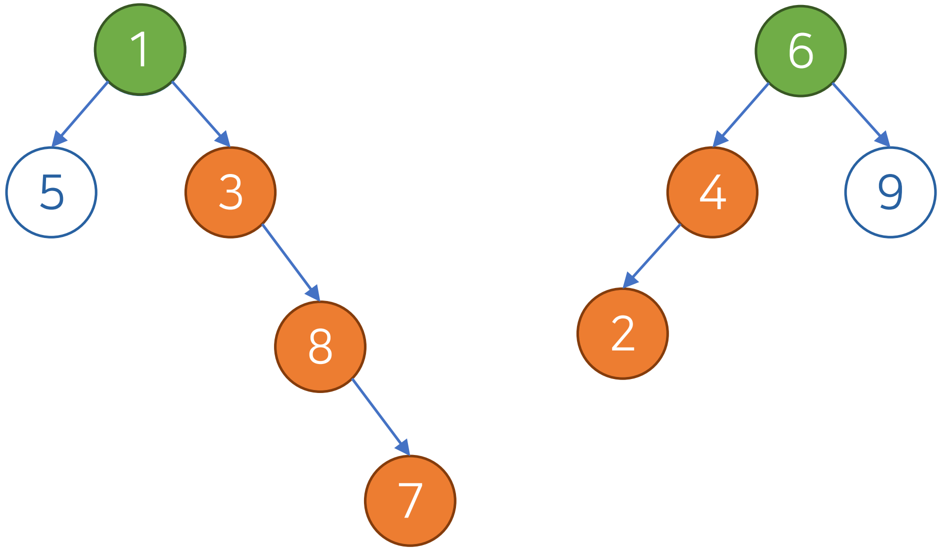
Node	1	2	3	4	5	6	7	8	9
Parent	1	4	1	6	1	6	7	3	6



- Union(7, 2)
    1. Find(7), Find(2)

Disjoint Set – Union-Find

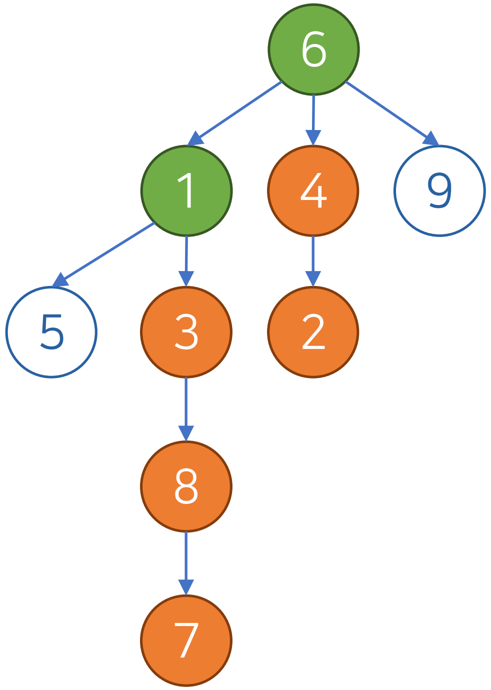
Node	1	2	3	4	5	6	7	8	9
Parent	1	4	1	6	1	6	7	3	6



- Union(7, 2)
  1. Find(7), Find(2)

Disjoint Set – Union-Find

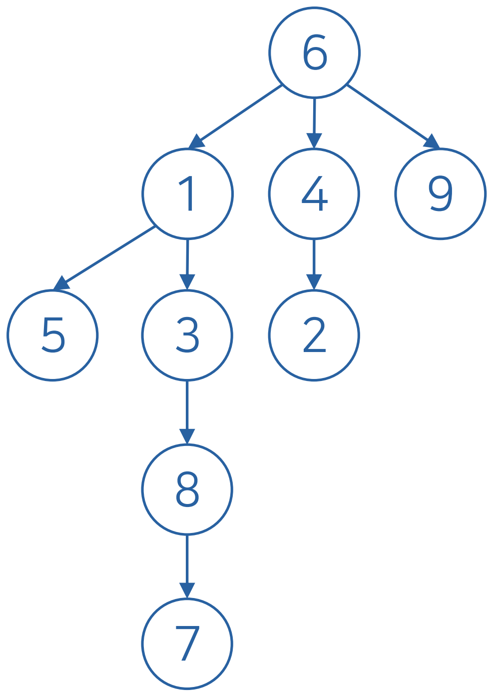
Node	1	2	3	4	5	6	7	8	9
Parent	1	4	1	6	1	6	7	3	6



- Union(7, 2)
  1. Find(7), Find(2)
  2. 합친다

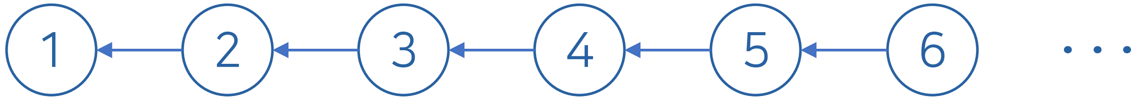
Disjoint Set – Union-Find

- 문제점
  - Union을 앞서 설명한대로 반복하면 Time Limit에 걸릴 수 있다.



Disjoint Set – Union-Find

- 문제점
  - Union을 앞서 설명한대로 반복하면 Time Limit에 걸릴 수 있다.
  - 아래와 같은 집합이 존재할 때, Union하고자 한다면 Find 연산에서 많은 시간이 소요될 수 있다.

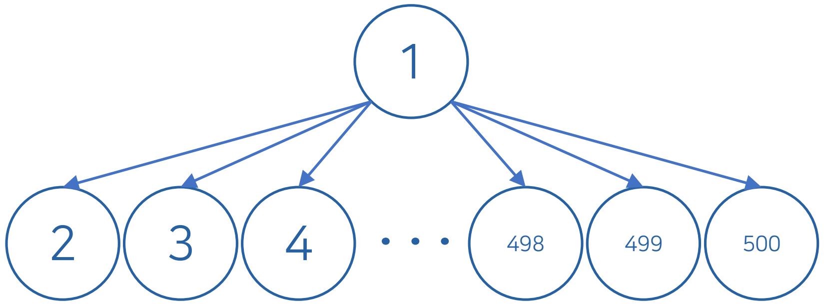


Disjoint Set – Union-Find

- 최적화
  - Tree가 균형을 이루지 않고 한쪽에만 치우쳐 있으면 Union시 Find 연산을 할 때 많은 시간이 소요된다.
  - 해결책
    - Path Compression Algorithm
    - Tree의 Node의 수를 비교한다.

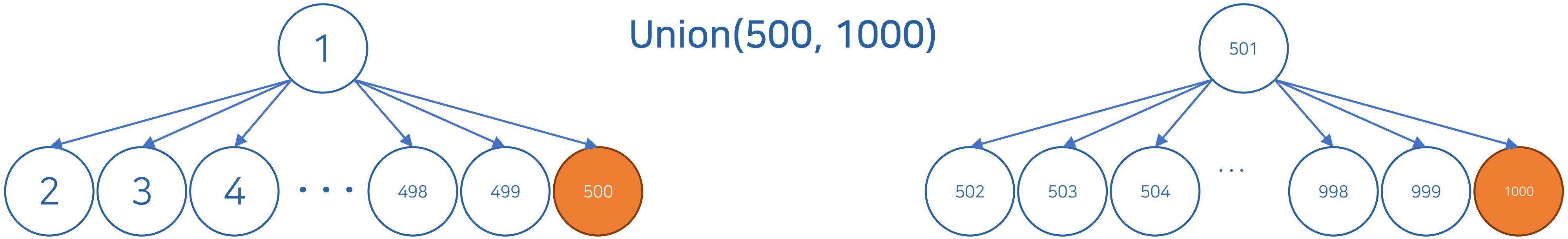
Disjoint Set – Union-Find

- Path Compression (경로 압축)
  - Find 연산은 단순히 어떤 원소가 어느 집합에 속하고 있는지 알려준다.
    - Root Node만 찾으면 된다
  - 만약 어떤 집합의 모든 Node의 Depth가 1이라면?



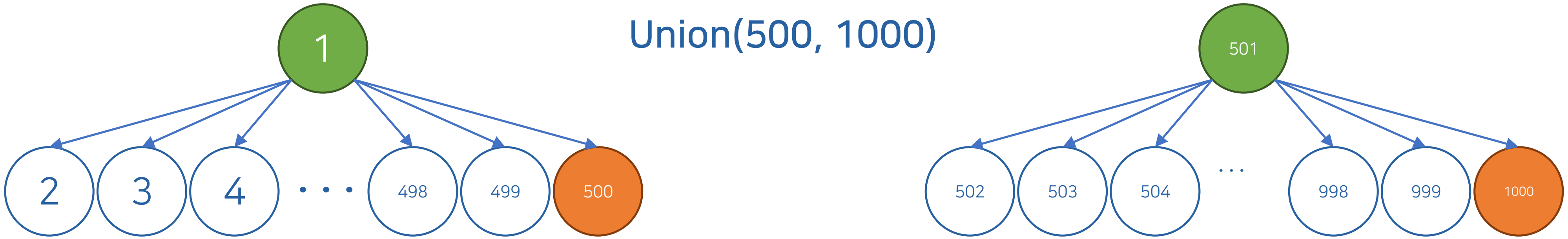
Disjoint Set – Union-Find

- Path Compression (경로 압축)
  - Find 연산은 단순히 어떤 원소가 어느 집합에 속하고 있는지 알려준다.
    - Root Node만 찾으면 된다
  - 만약 어떤 집합의 모든 Node의 Depth가 1이라면?

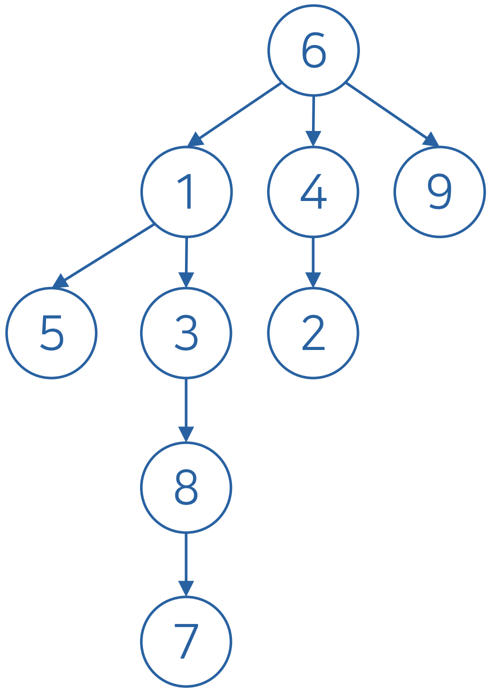


Disjoint Set – Union-Find

- Path Compression (경로 압축)
  - Find 연산은 단순히 어떤 원소가 어느 집합에 속하고 있는지 알려준다.
    - Root Node만 찾으면 된다
  - 만약 어떤 집합의 모든 Node의 Depth가 1이라면?

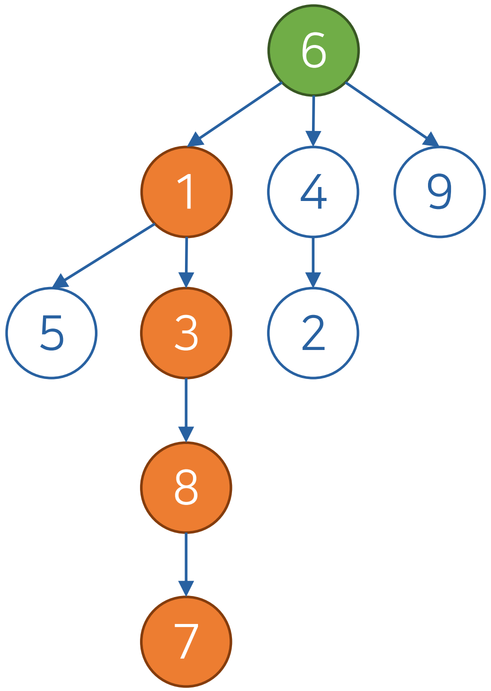


Disjoint Set – Union-Find



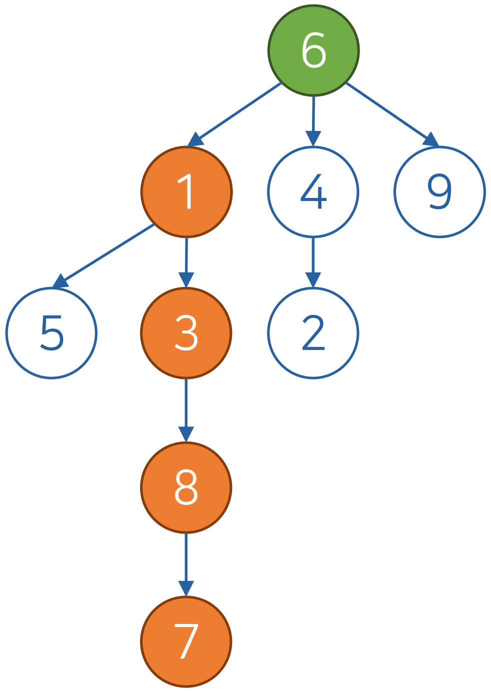
- Path Compression (경로 압축)
  1. Union시 Root Node를 찾아 방문했던 모든 Node들을 Root Node의 Child Node로 만든다.

Disjoint Set – Union-Find

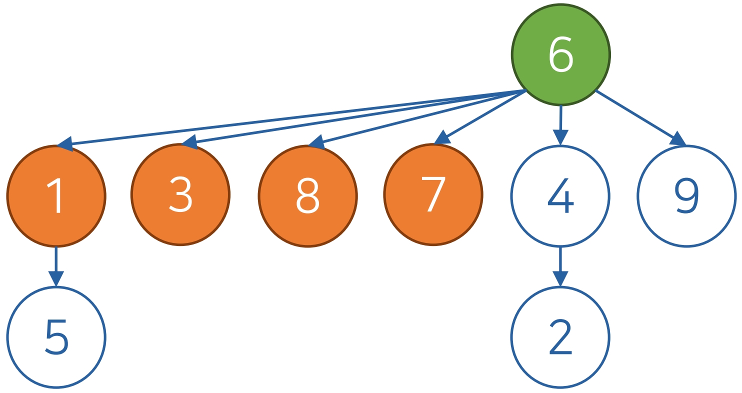


- Path Compression (경로 압축)
  1. Union시 Root Node를 찾아 방문했던 모든 Node들을 Root Node의 Child Node로 만든다.

Disjoint Set – Union-Find

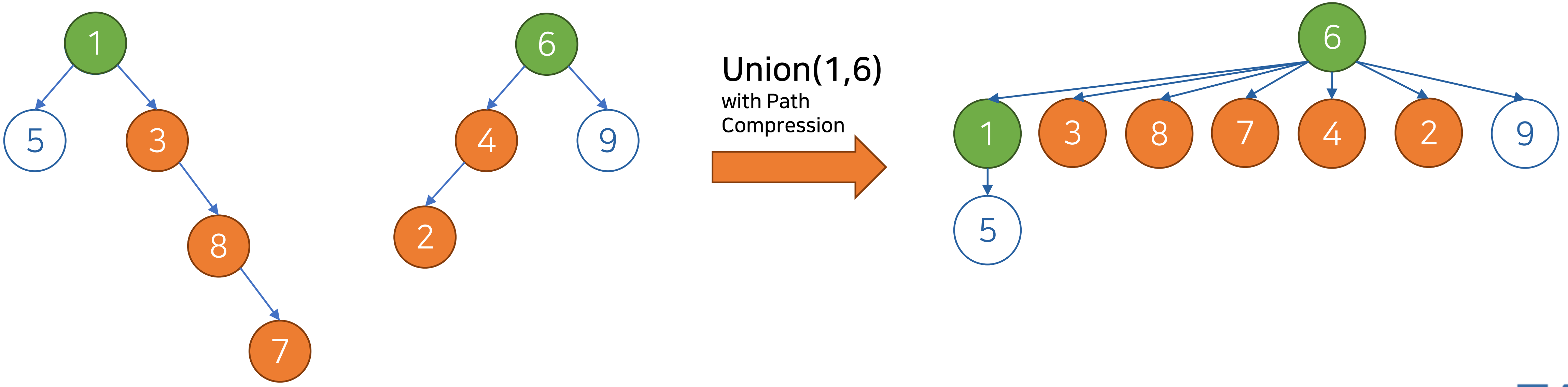


- Path Compression (경로 압축)
  1. Union시 Root Node를 찾아 방문했던 모든 Node들을 Root Node의 Child Node로 만든다.



Disjoint Set – Union-Find

- Path Compression (경로 압축)
  - Union(7, 2) → 1과 6을 합친다.



Disjoint Set

BOJ 1717 집합의 표현
BOJ 4195 친구 네트워크

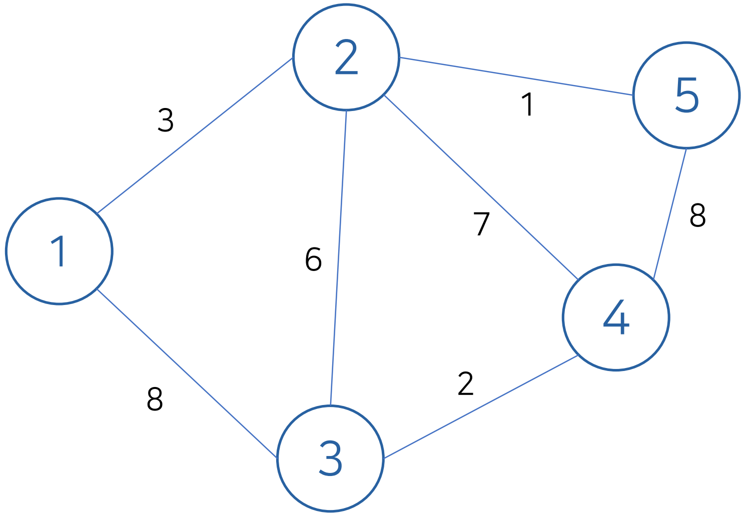
Disjoint Set

BOJ 13306 트리

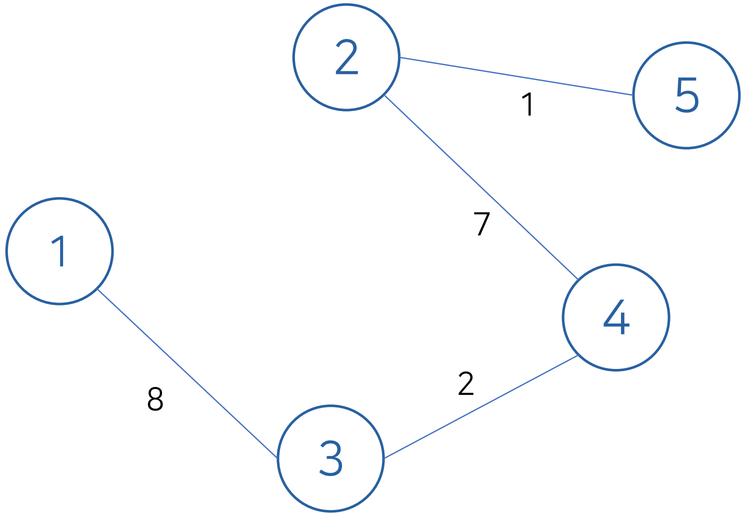
MCST

Spanning Tree

- 신장 트리 (Spanning Tree)
  - Graph의 모든 Vertex를 포함하는 Sub Graph이자 Tree
  - 신장 트리의 Edge의 개수는 N-1개이다. (N : Node의 수)



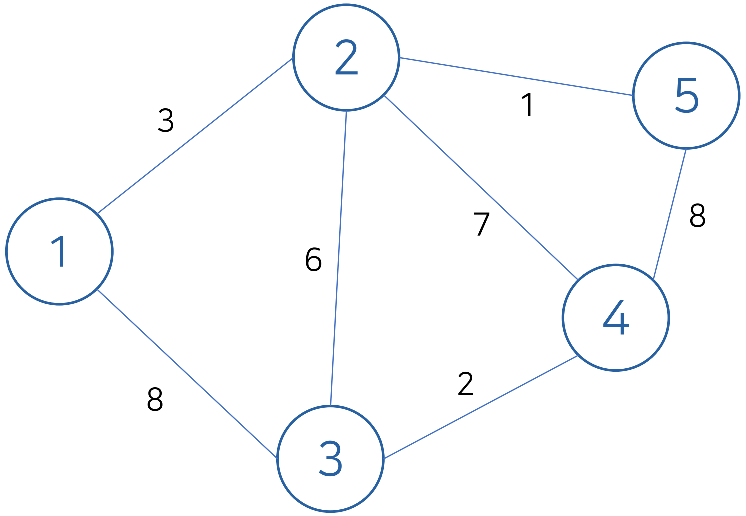
Weight Graph



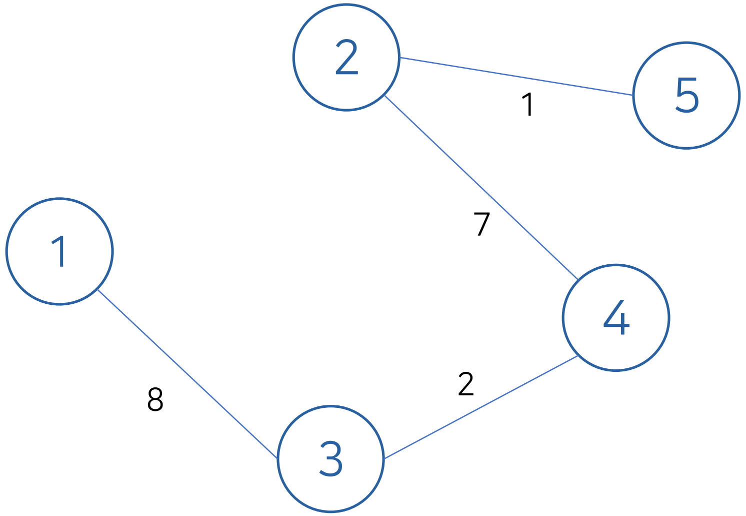
Spanning Tree

Spanning Tree

- 신장 트리 (Spanning Tree)
  - Graph의 모든 Vertex를 포함하는 Sub Graph이자 Tree
  - 신장 트리의 Edge의 개수는 N-1개이다. (N : Node의 수)



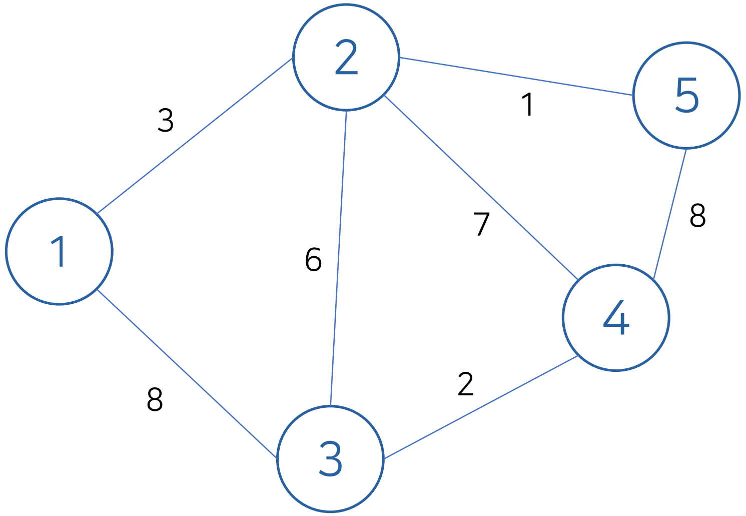
Weight Graph



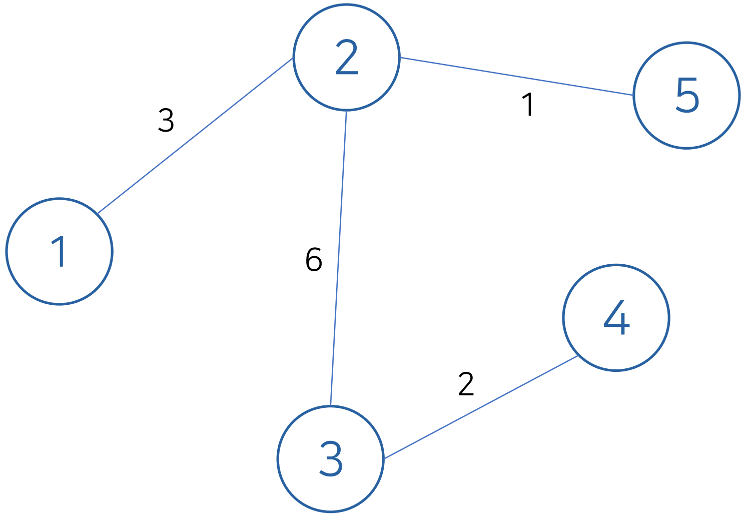
Spanning Tree

MCST

- 최소 비용 신장 트리 (Minimum Cost Spanning Tree)
  - Graph의 신장 트리 중 Edge의 가중치 합이 최소인 신장 트리



Weight Graph



MCST

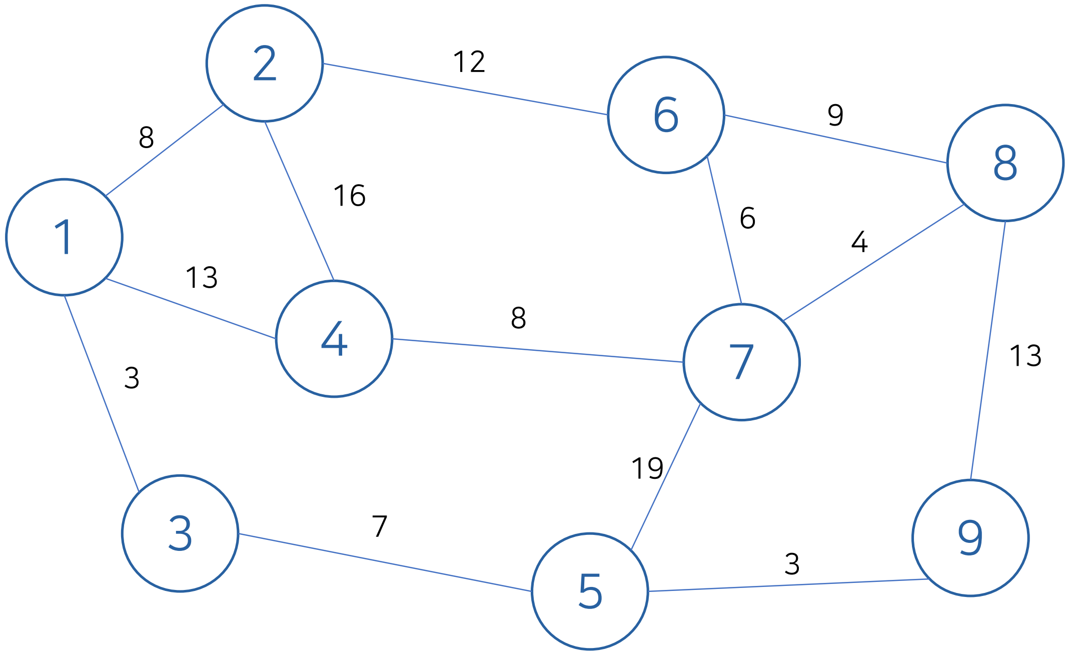
MCST

- 최소 비용 신장 트리 (Minimum Cost Spanning Tree)
  - Kruskal's Algorithm
  - Prim's Algorithm

MCST – Kruskal's Algorithm

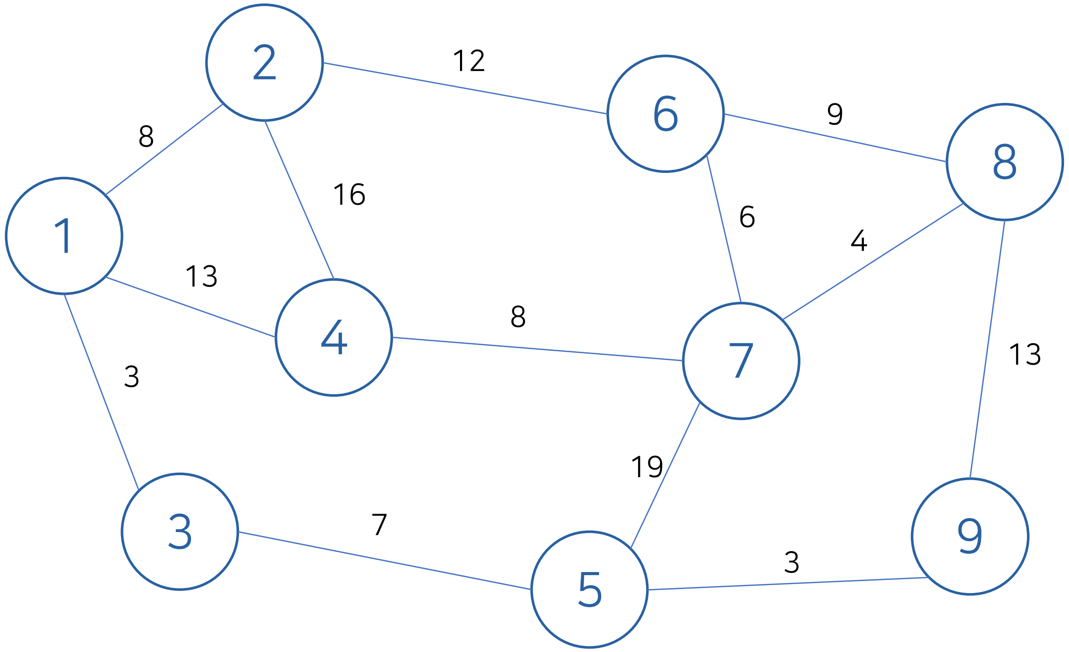
- Kruskal's Algorithm
  1. Graph의 Edge들을 가중치 기준으로 오름차순 정렬한다.,
  2. 가중치가 작은 Edge들 부터 시작하여 다음을 반복한다.
     1. 선택된 Edge의 양 끝에 있는 Node들이 서로 다른 집합에 있는지 확인한다. (Find)
     2. 만약 같은 집합에 속하고 있지 않는다면, 두 Node가 속한 집합을 합친다. (Union)
        하지만 같은 집합에 속하고 있다면, 1로 넘어간다.
     3. 선택된 Edge를 MCST의 Edge로 취급한다.

MCST – Kruskal's Algorithm



MCST – Kruskal's Algorithm

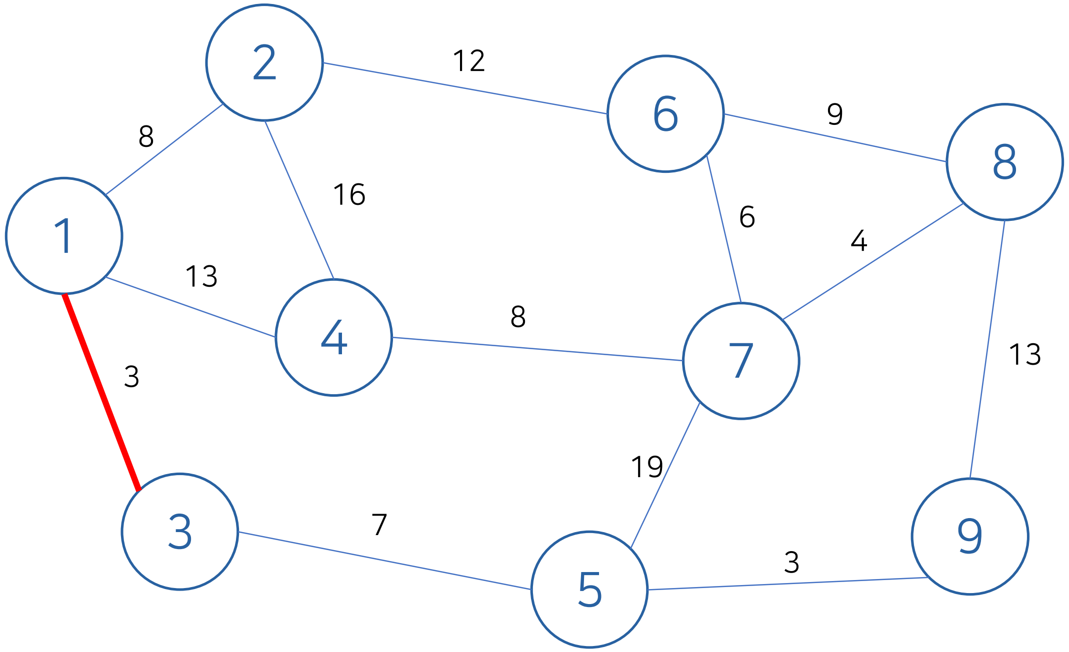
- Edge의 가중치를 기준으로 정렬



U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

MCST – Kruskal’s Algorithm

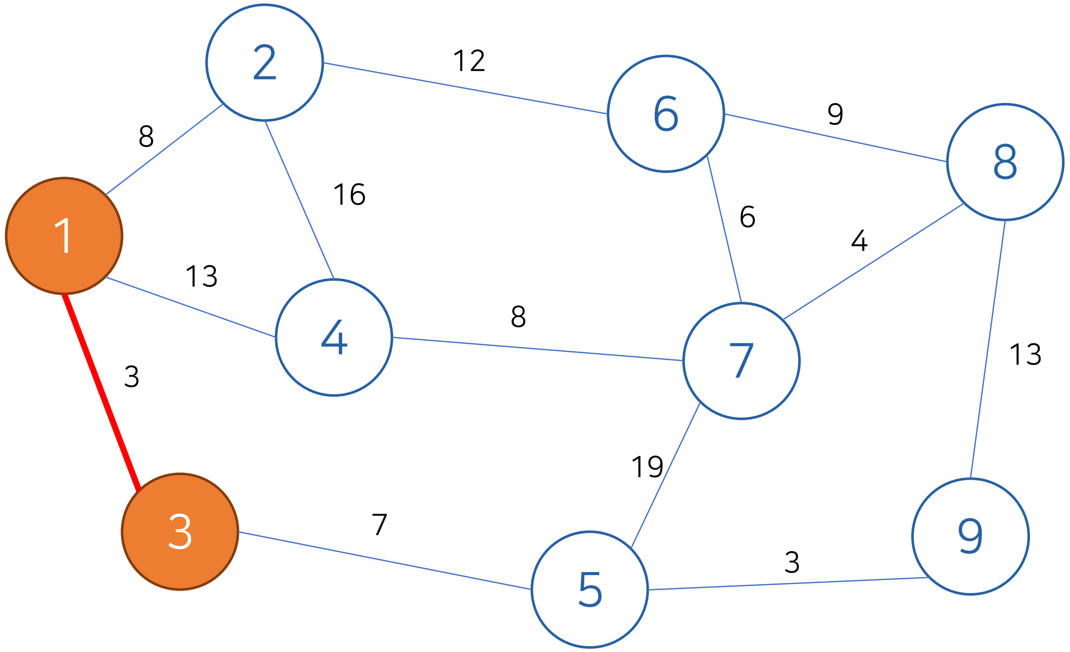
- Edge의 가중치가 가장 작은 것부터



U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

MCST – Kruskal's Algorithm

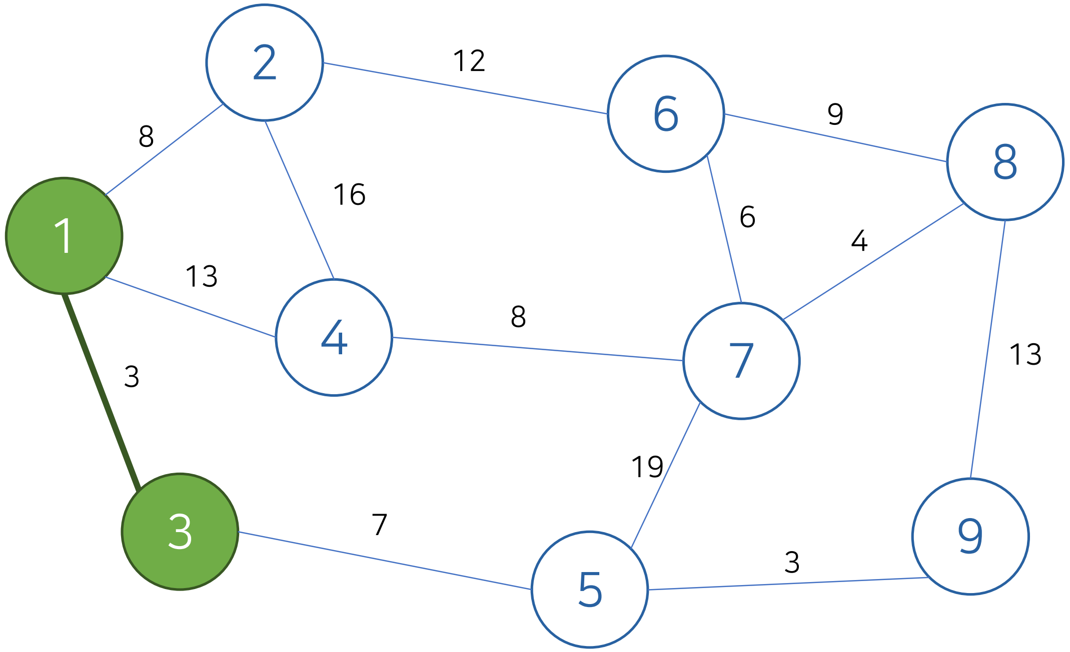
- Edge의 가중치가 가장 작은 것부터
  1. 선택된 Edge의 양 끝 Node들이 서로 같은 집합에 있는지 확인한다. (Find)



U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

MCST – Kruskal's Algorithm

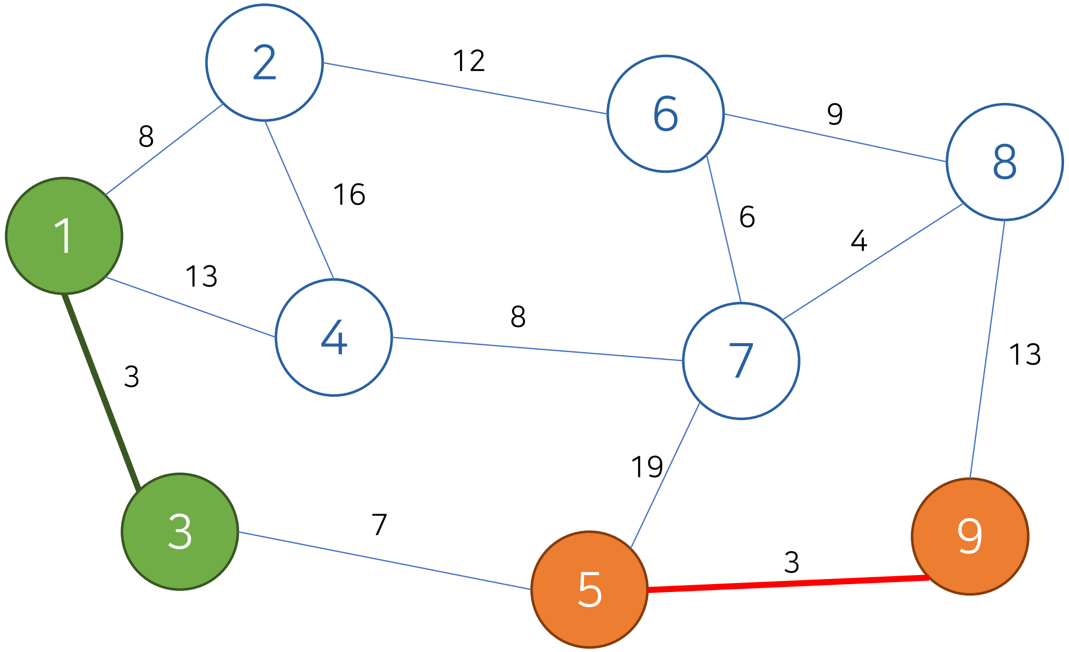
- Edge의 가중치가 가장 작은 것부터
    2. 만약 같은 집합에 속하고 있지 않는다면 두 Node가 속한 집합을 합친다. (Union)



U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

MCST – Kruskal's Algorithm

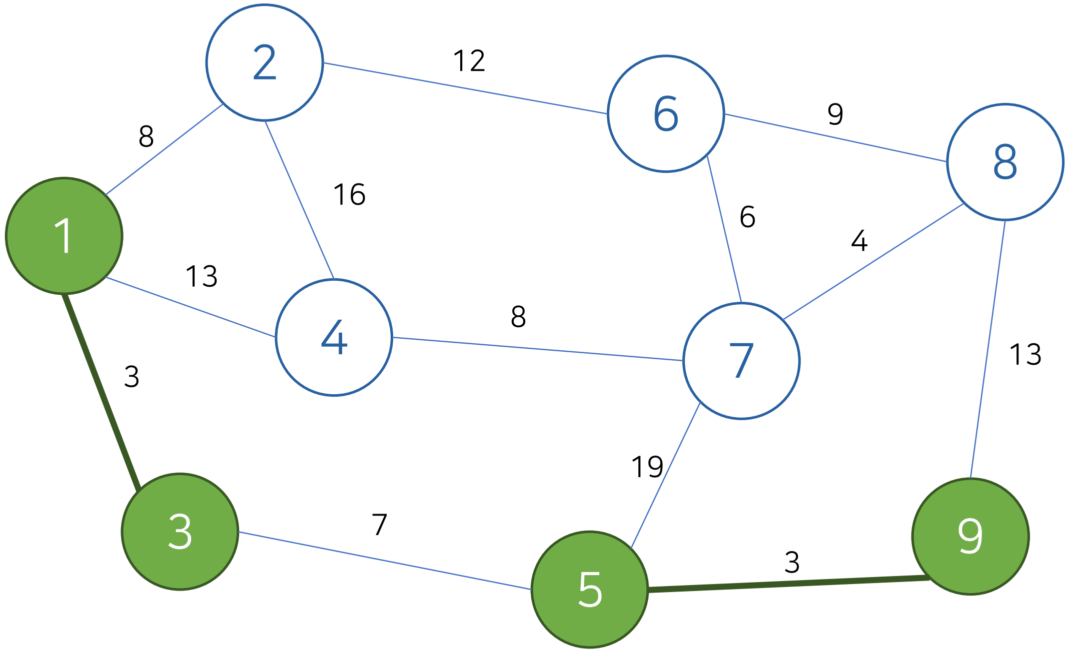
- Edge의 가중치가 가장 작은 것부터
  1. 선택된 Edge의 양 끝 Node들이 서로 같은 집합에 있는지 확인한다. (Find)



U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

MCST – Kruskal's Algorithm

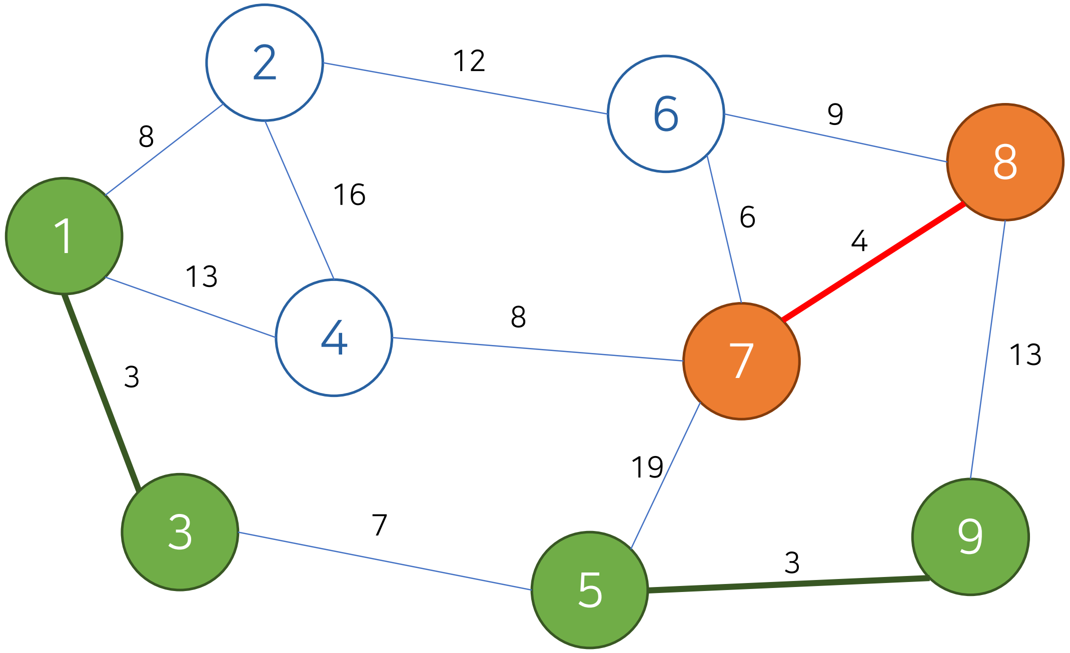
- Edge의 가중치가 가장 작은 것부터
  2. 만약 같은 집합에 속하고 있지 않는다면 두 Node가 속한 집합을 합친다. (Union)



U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

MCST – Kruskal's Algorithm

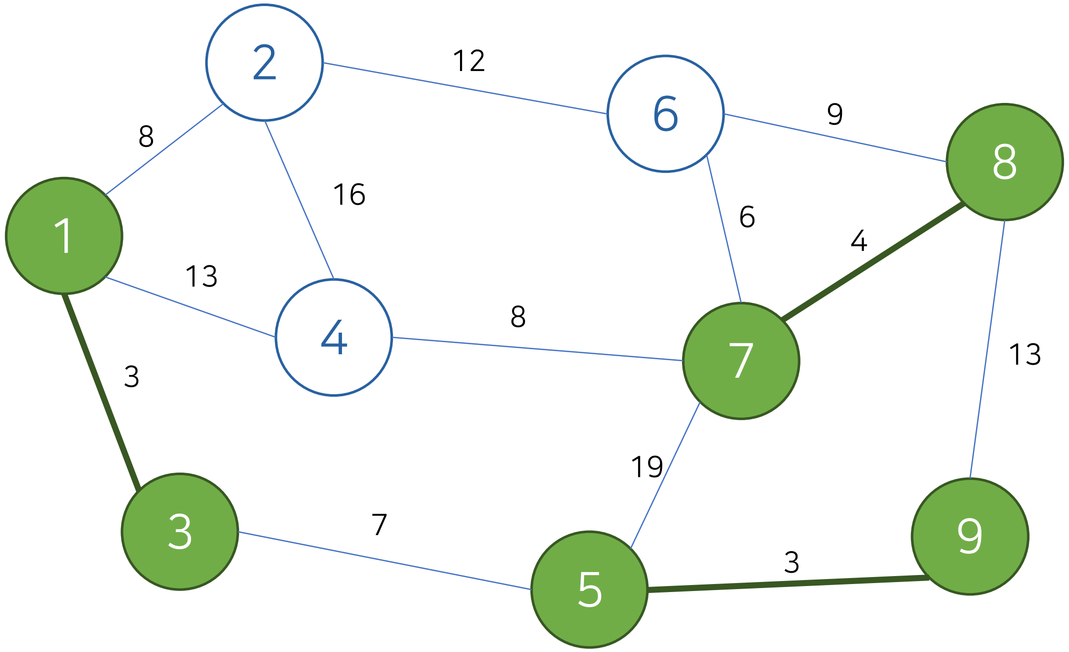
- Edge의 가중치가 가장 작은 것부터
  1. 선택된 Edge의 양 끝 Node들이 서로 같은 집합에 있는지 확인한다. (Find)



U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

MCST – Kruskal's Algorithm

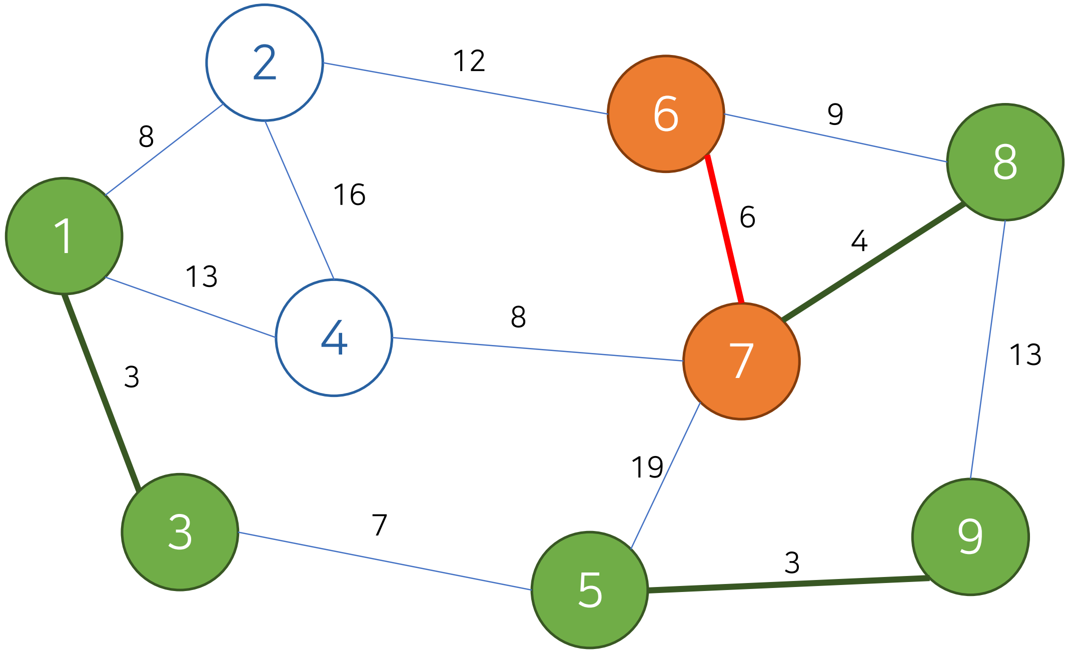
- Edge의 가중치가 가장 작은 것부터
    2. 만약 같은 집합에 속하고 있지 않는다면 두 Node가 속한 집합을 합친다. (Union)



U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

MCST – Kruskal’s Algorithm

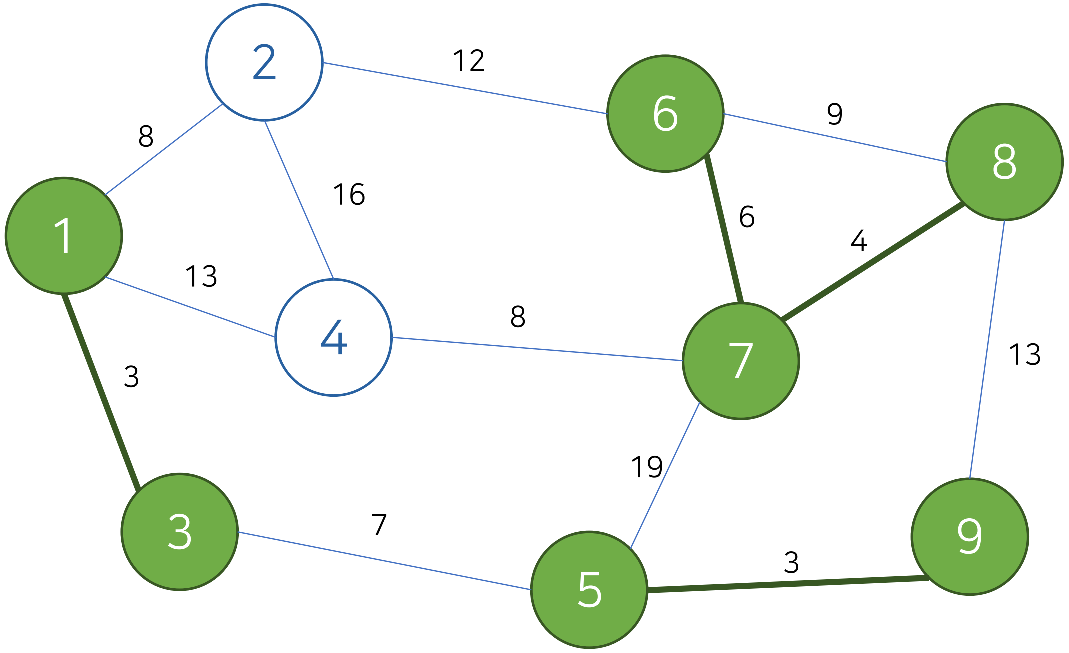
- Edge의 가중치가 가장 작은 것부터
  1. 선택된 Edge의 양 끝 Node들이 서로 같은 집합에 있는지 확인한다. (Find)



U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

MCST – Kruskal's Algorithm

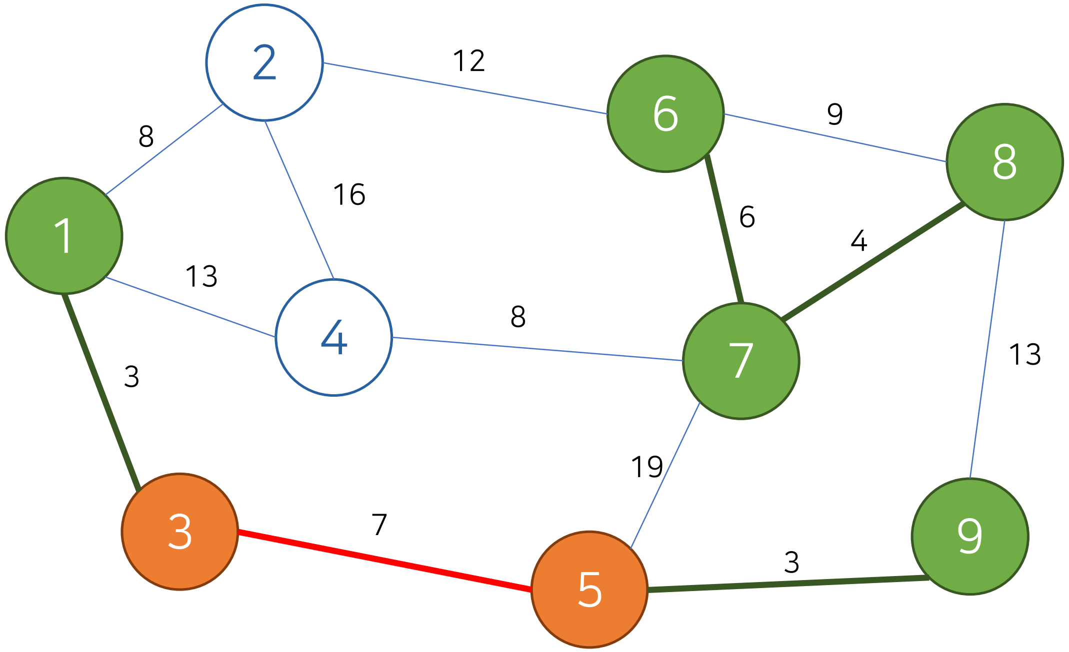
- Edge의 가중치가 가장 작은 것부터
  2. 만약 같은 집합에 속하고 있지 않는다면 두 Node가 속한 집합을 합친다. (Union)



U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

MCST – Kruskal's Algorithm

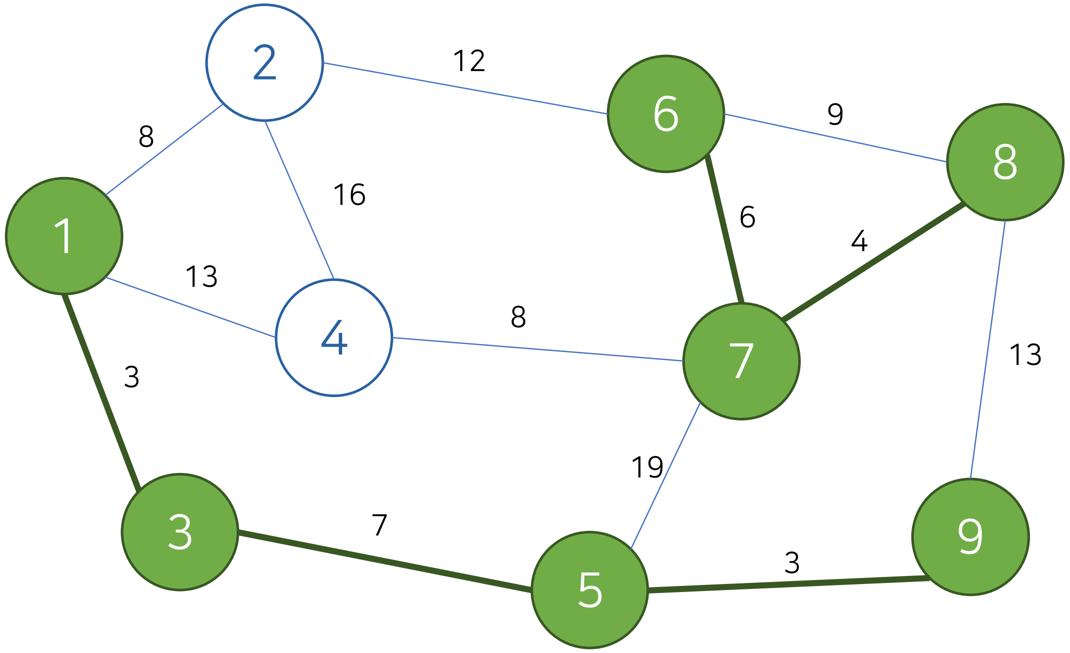
- Edge의 가중치가 가장 작은 것부터
  1. 선택된 Edge의 양 끝 Node들이 서로 같은 집합에 있는지 확인한다. (Find)



U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

MCST – Kruskal's Algorithm

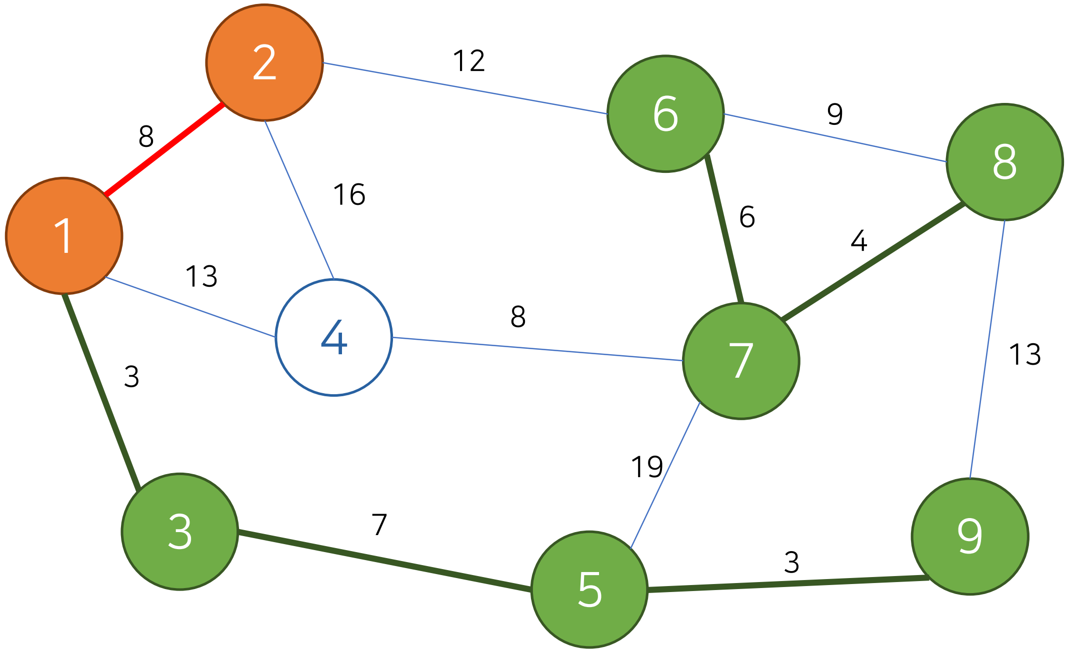
- Edge의 가중치가 가장 작은 것부터
  2. 만약 같은 집합에 속하고 있지 않는다면 두 Node가 속한 집합을 합친다. (Union)



U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

MCST – Kruskal's Algorithm

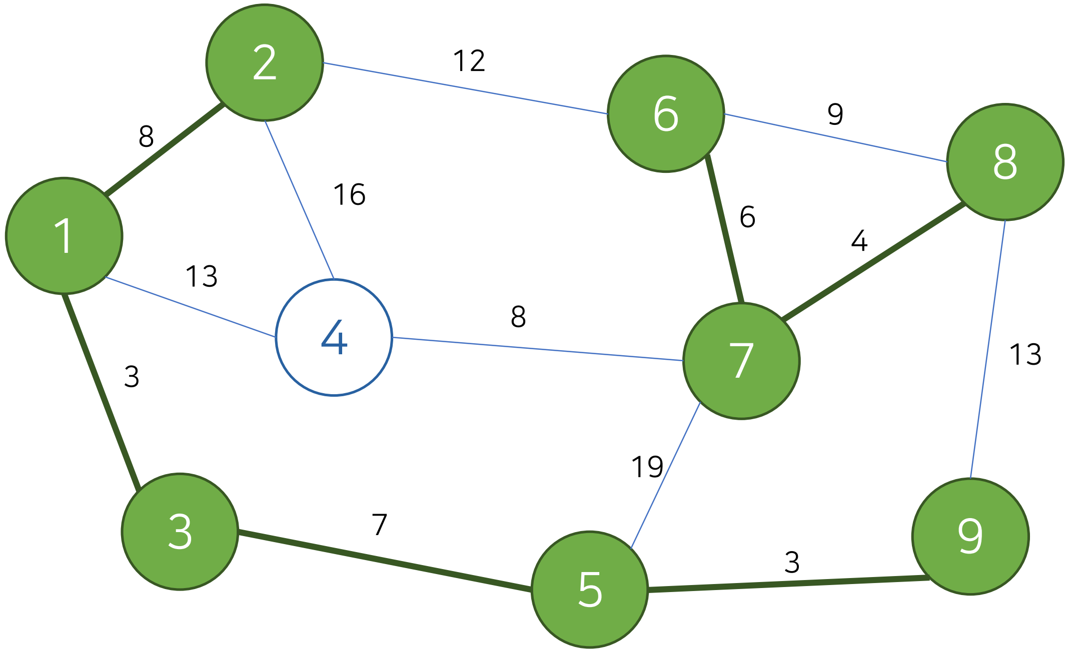
- Edge의 가중치가 가장 작은 것부터
  1. 선택된 Edge의 양 끝 Node들이 서로 같은 집합에 있는지 확인한다. (Find)



U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

MCST – Kruskal's Algorithm

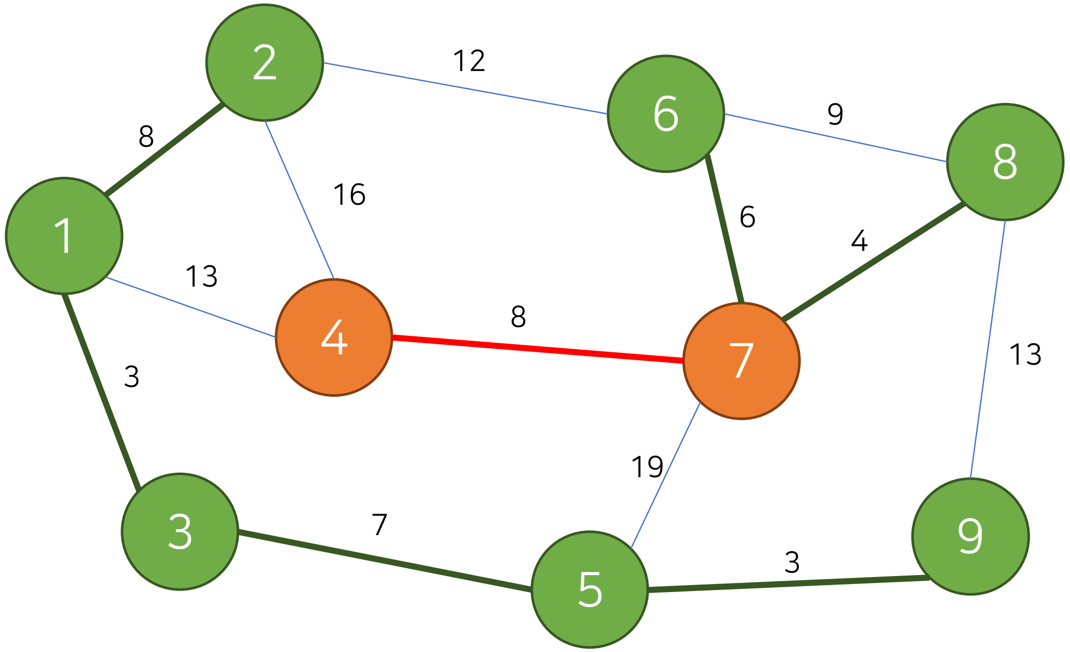
- Edge의 가중치가 가장 작은 것부터
  2. 만약 같은 집합에 속하고 있지 않는다면 두 Node가 속한 집합을 합친다. (Union)



U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

MCST – Kruskal's Algorithm

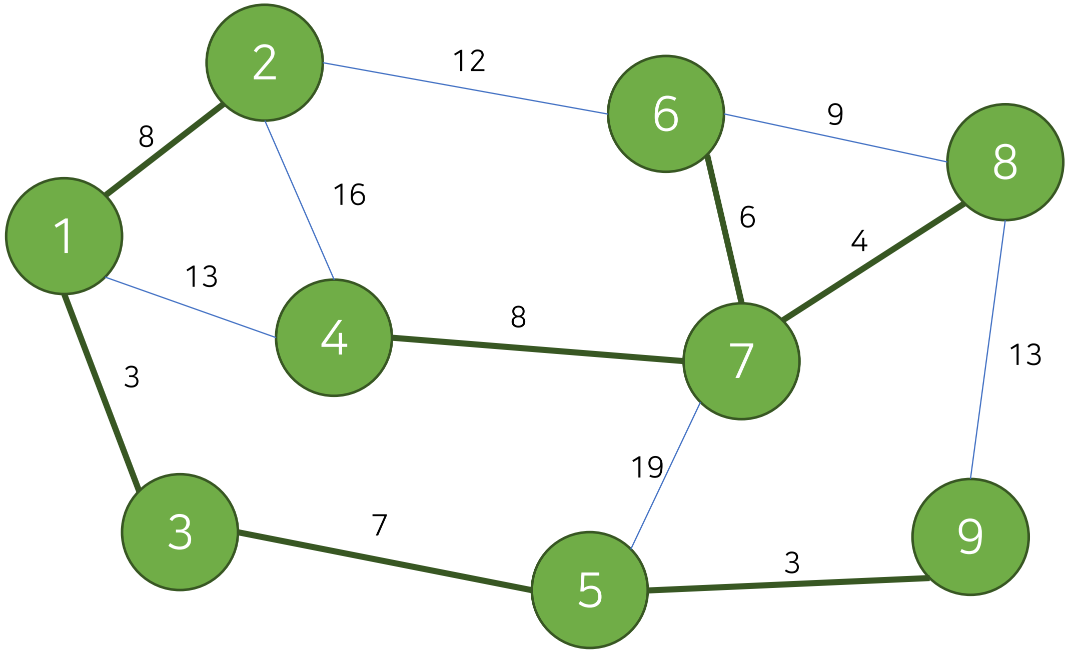
- Edge의 가중치가 가장 작은 것부터
  1. 선택된 Edge의 양 끝 Node들이 서로 같은 집합에 있는지 확인한다. (Find)



U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

MCST – Kruskal's Algorithm

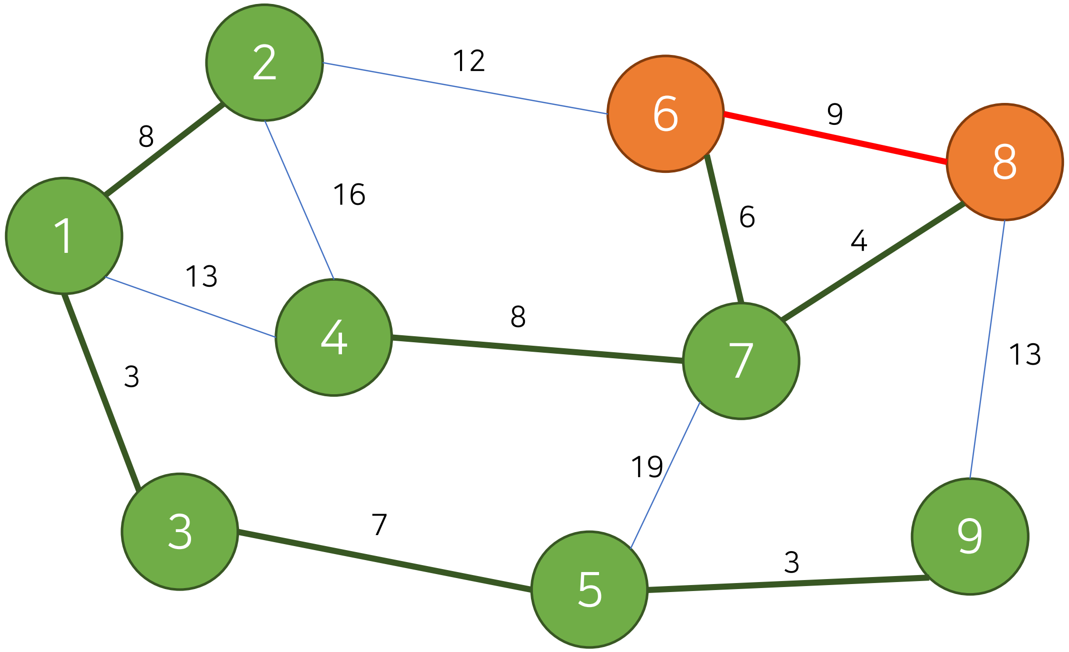
- Edge의 가중치가 가장 작은 것부터
  2. 만약 같은 집합에 속하고 있지 않는다면 두 Node가 속한 집합을 합친다. (Union)



U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

MCST – Kruskal's Algorithm

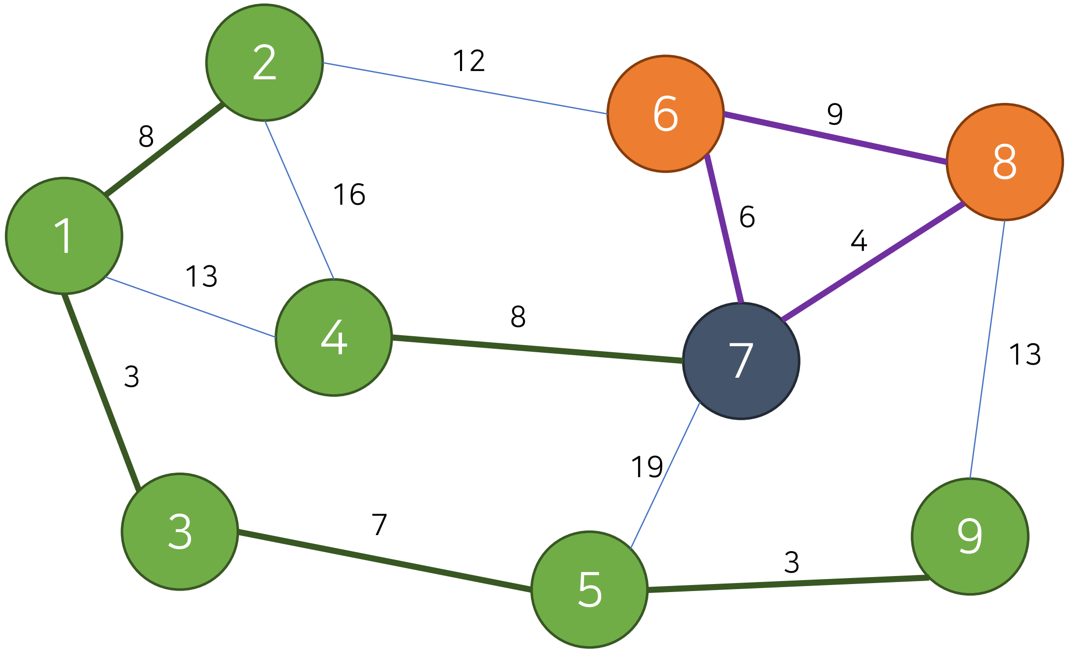
- Edge의 가중치가 가장 작은 것부터
  1. 선택된 Edge의 양 끝 Node들이 서로 같은 집합에 있는지 확인한다. (Find)



U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

MCST – Kruskal's Algorithm

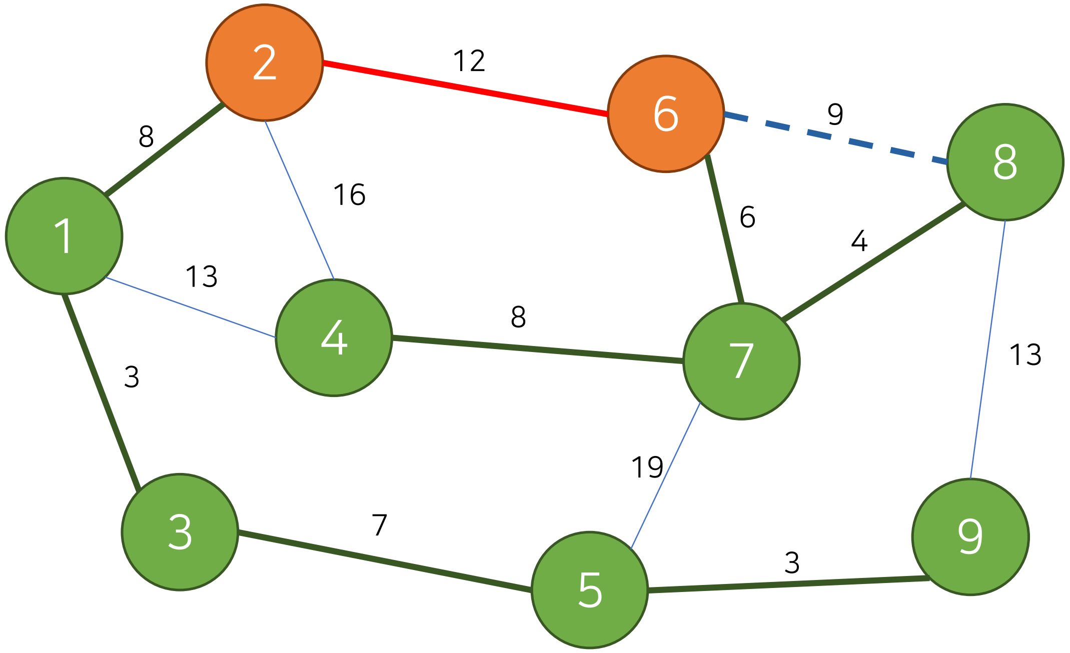
- Edge의 가중치가 가장 작은 것부터
  3. 만약 같은 집합에 속한다면 넘어간다.



U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

MCST – Kruskal's Algorithm

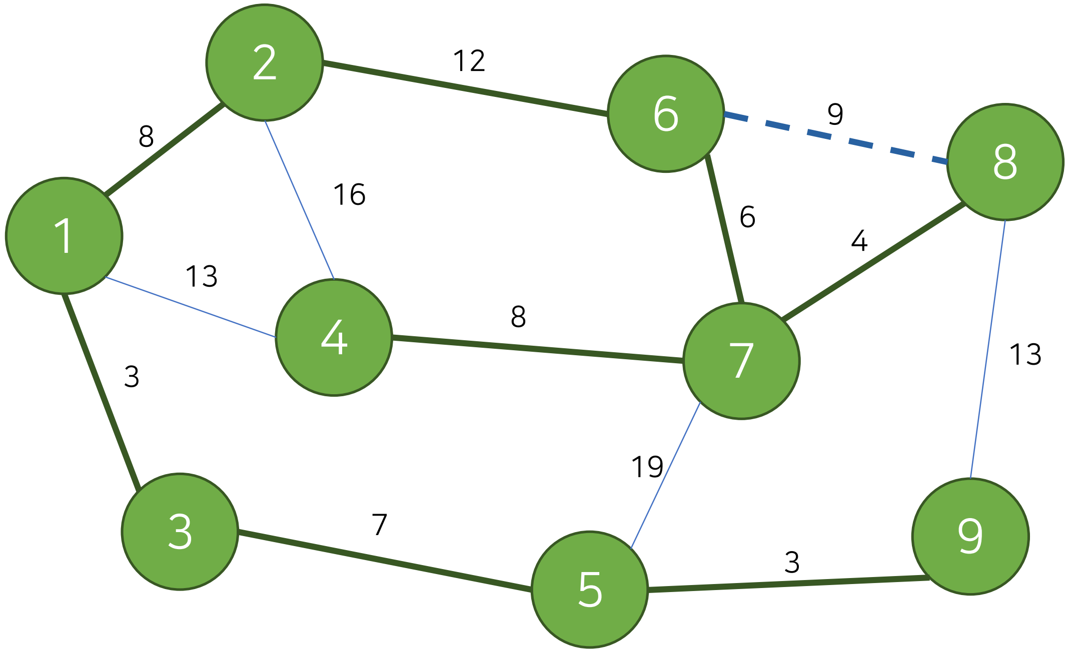
- Edge의 가중치가 가장 작은 것부터
  1. 선택된 Edge의 양 끝 Node들이 서로 같은 집합에 있는지 확인한다. (Find)



U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

MCST – Kruskal's Algorithm

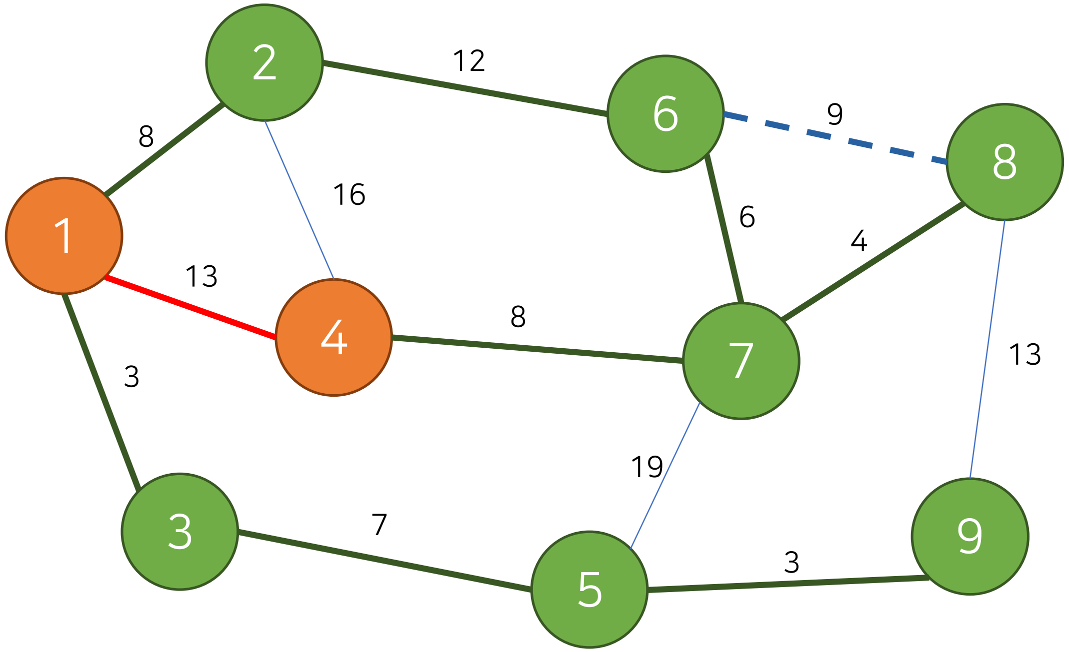
- Edge의 가중치가 가장 작은 것부터
  2. 만약 같은 집합에 속하고 있지 않는다면 두 Node가 속한 집합을 합친다. (Union)



U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

MCST – Kruskal's Algorithm

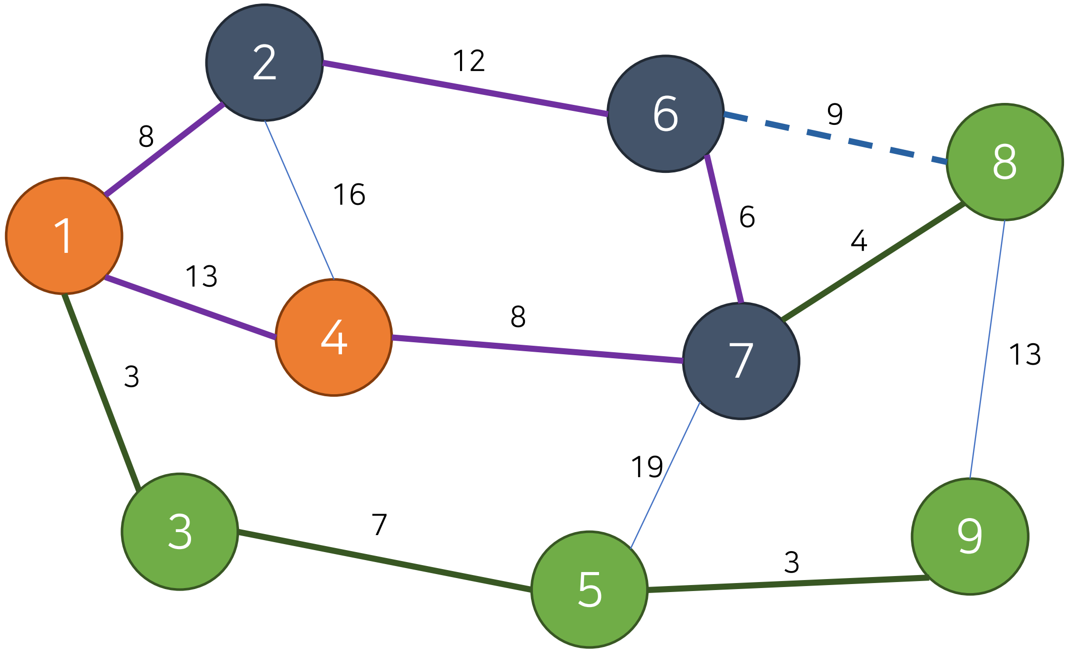
- Edge의 가중치가 가장 작은 것부터
  1. 선택된 Edge의 양 끝 Node들이 서로 같은 집합에 있는지 확인한다. (Find)



U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

MCST – Kruskal's Algorithm

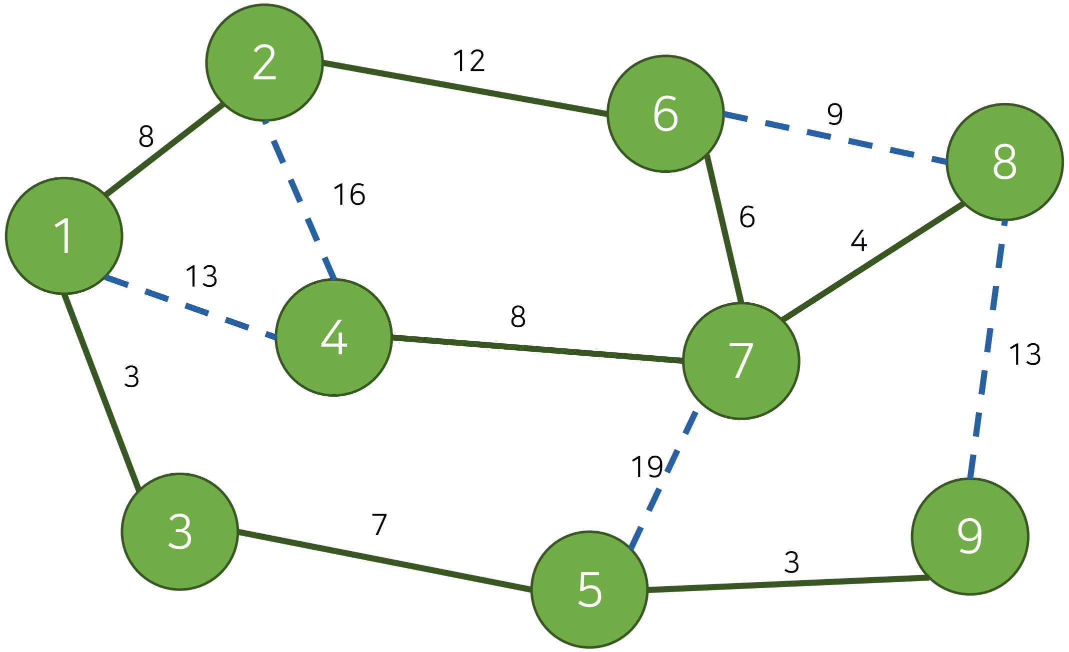
- Edge의 가중치가 가장 작은 것부터
  3. 만약 같은 집합에 속한다면 넘어간다.



U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

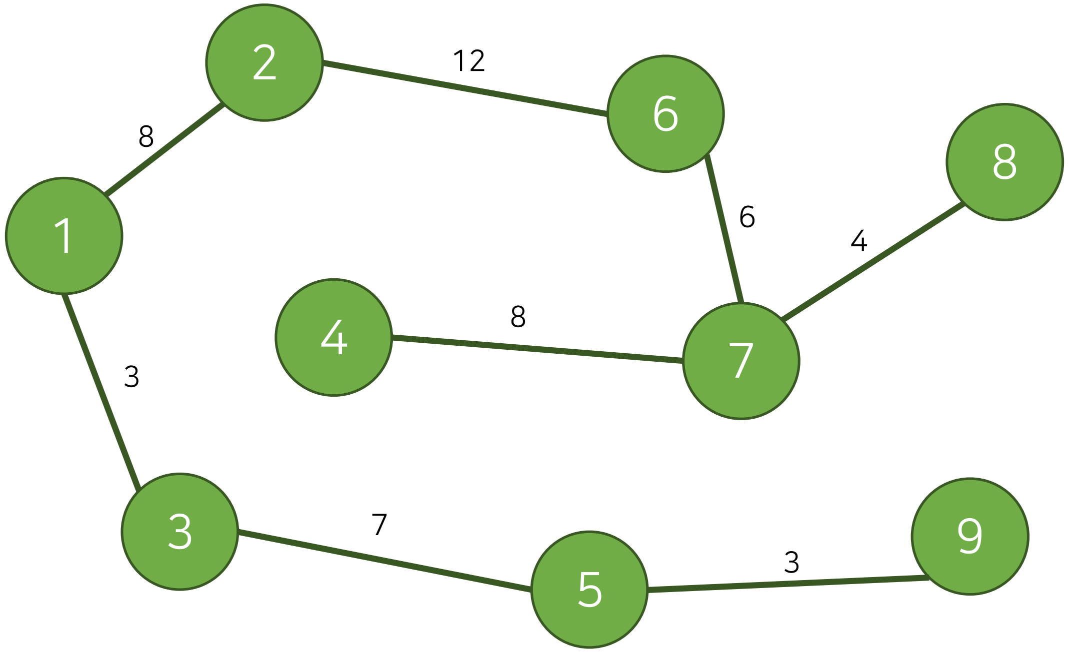
MCST – Kruskal's Algorithm

- Edge의 가중치가 가장 작은 것부터
  3. 만약 같은 집합에 속한다면 넘어간다.



U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

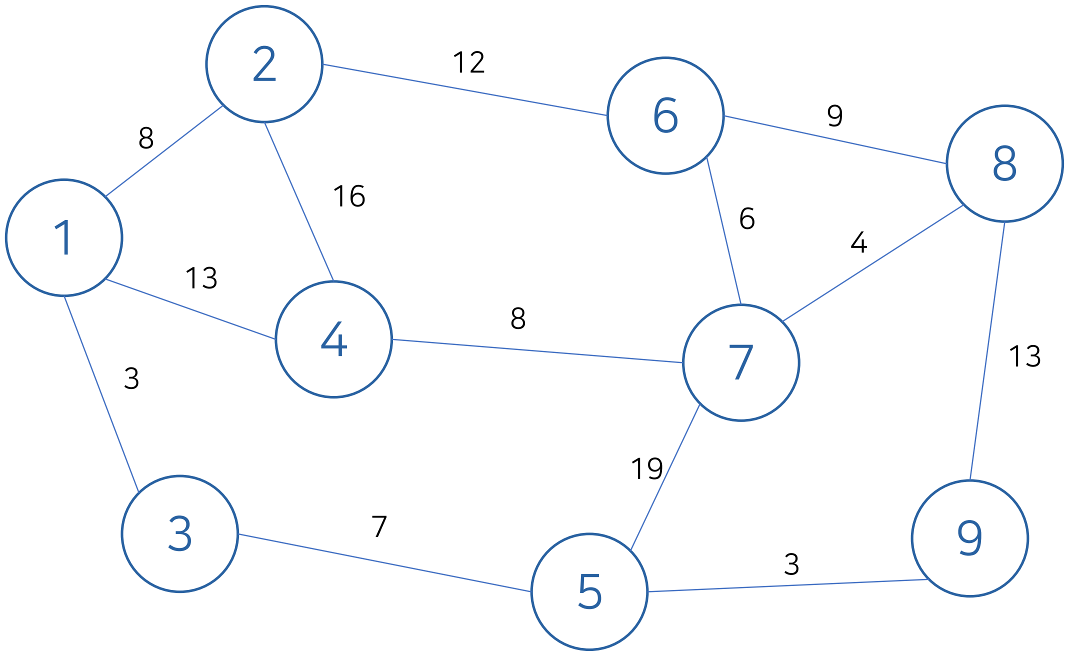
MCST – Kruskal's Algorithm



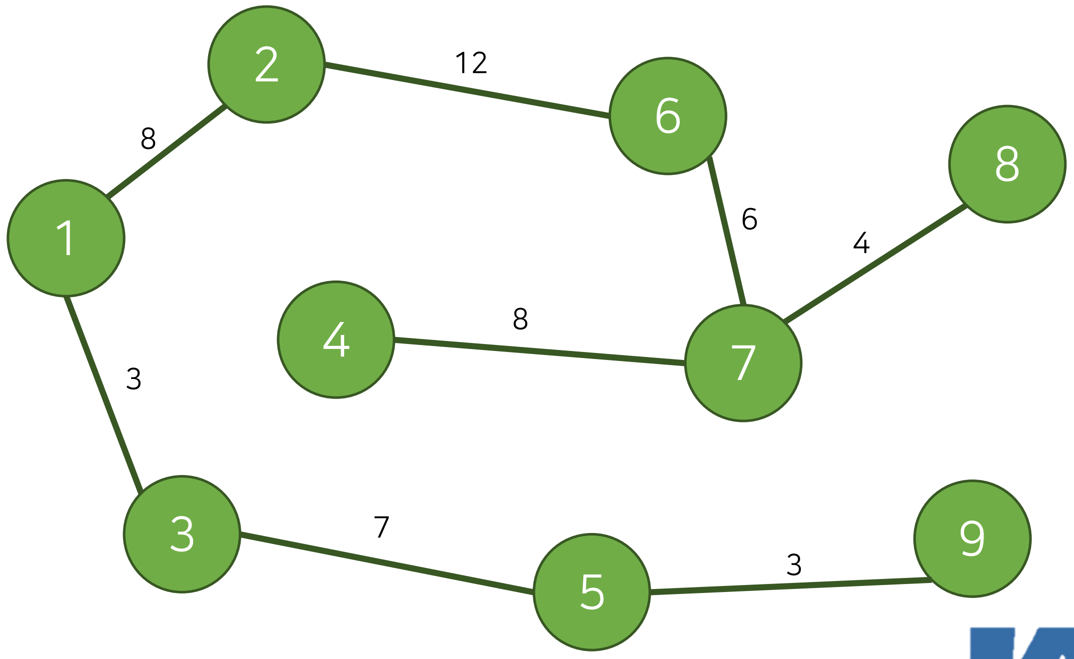
U	V	Weight
1	3	3
5	9	3
7	8	4
6	7	6
3	5	7
1	2	8
4	7	8
6	8	9
2	6	12
1	4	13
8	9	13
2	4	16
5	7	19

MCST – Kruskal's Algorithm

- Minimum Cost = 51



Weighted Graph



MCST

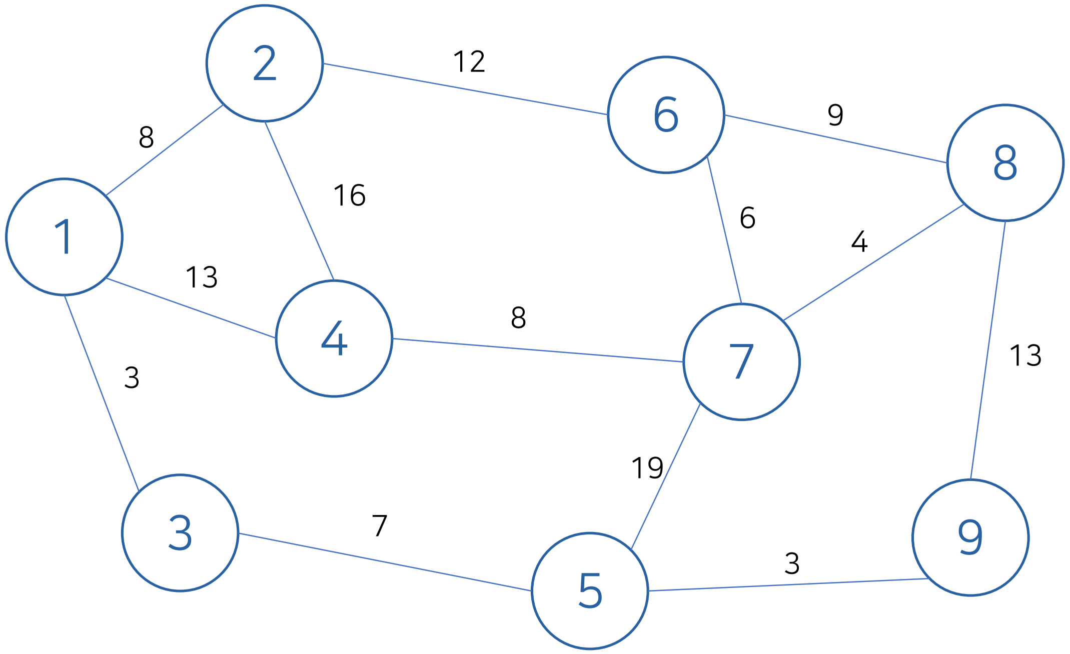
MCST – Kruskal's Algorithm

- Time Complexity
  - $O(E\log V)$

MCST – Prim's Algorithm

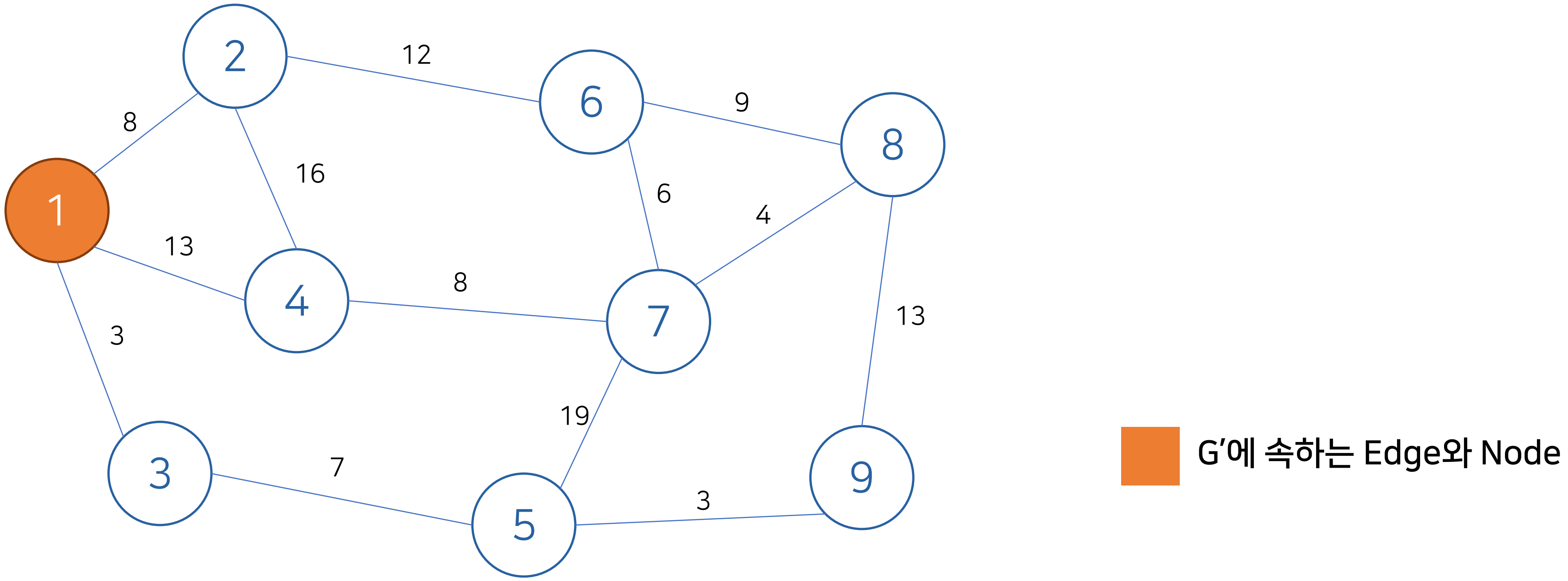
- Prim's Algorithm
  1. 임의의 Node를 선택하고 또 다른 Graph G'에 추가한다. (초기 G'는 빈 Graph)
  2. G'에서 연결된 Edge를 선택하는데, 그 간선의 양쪽 Node G'에 모두 속하지 않는 Edge 중 가중치의 값이 가장 작은 Edge을 선택한다.
  3. 그 Edge과 Edge에 연결된 Node를 G'에 추가한다.
  4. 모든 Node들이 G'에 속할 때 까지 2번과 3번을 반복한다.
  5. 모든 과정을 마치면 G'가 Minimum Cost Spanning Tree이다.

MCST – Prim's Algorithm



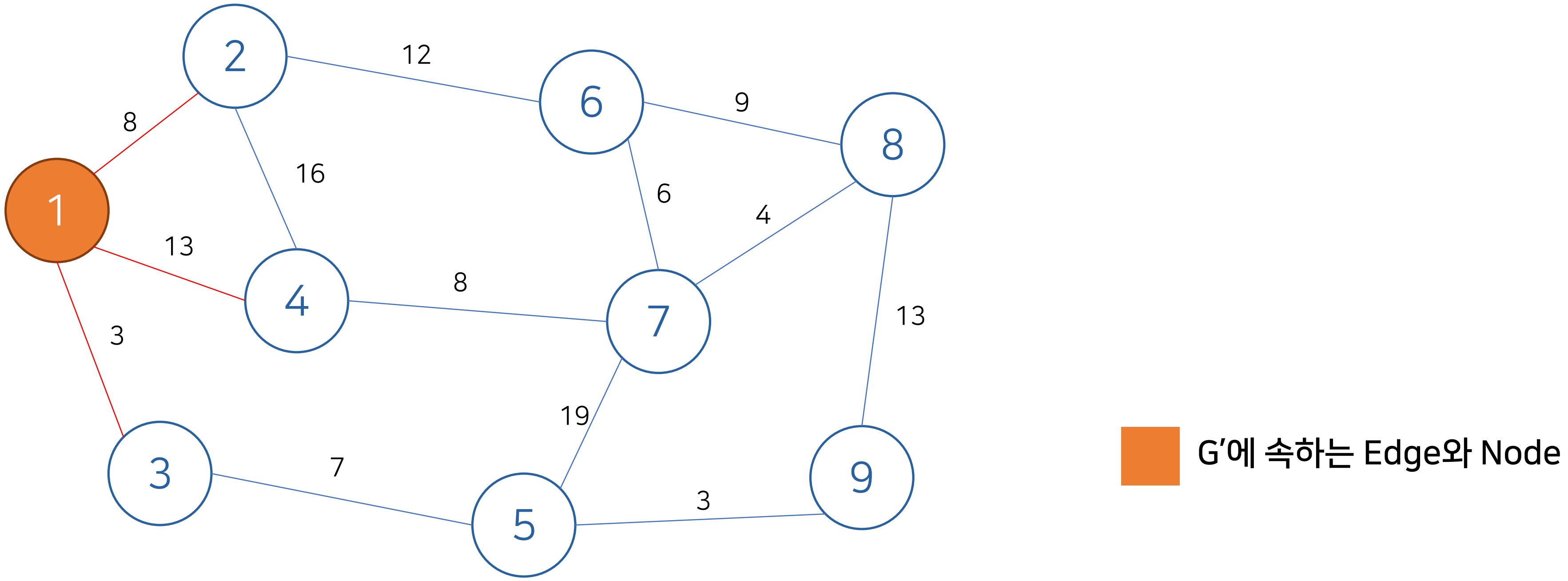
MCST – Prim's Algorithm

- 임의의 Node를 선택하고 또 다른 Graph G'에 추가한다.



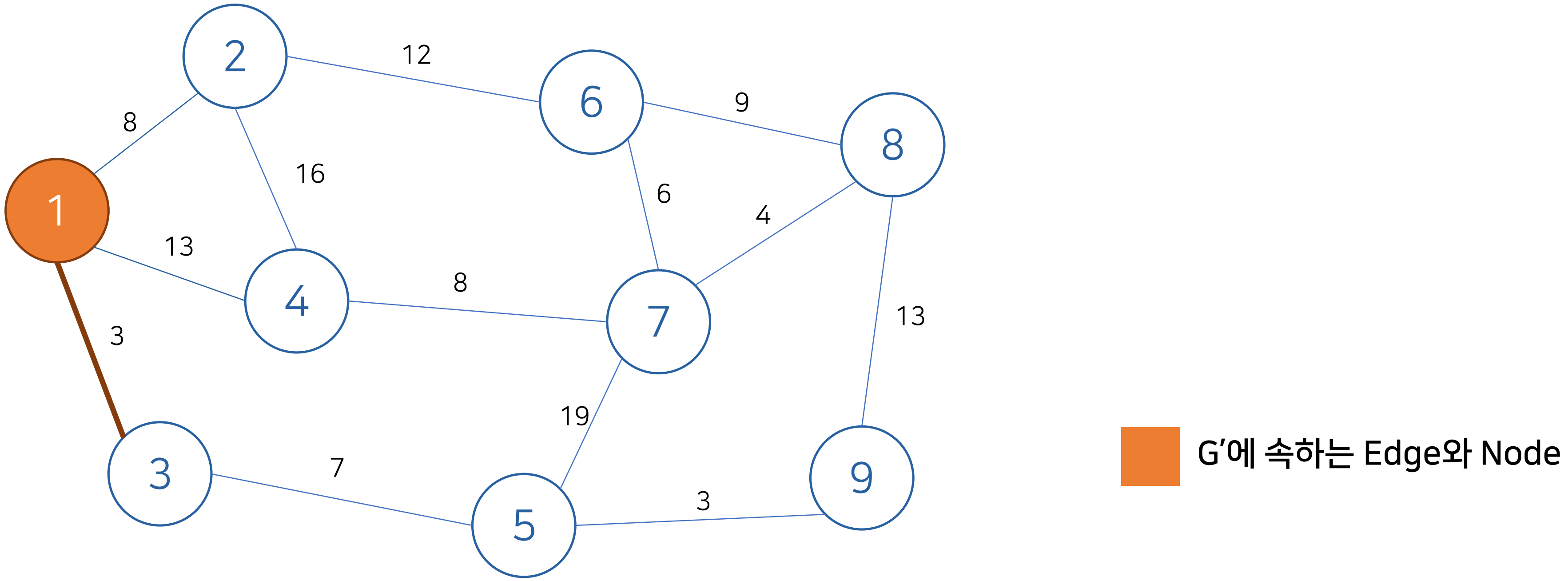
MCST – Prim's Algorithm

- G'에서 연결된 Edge를 선택하는데, 그 간선의 양쪽 Node가 G'에 모두 속하지 않는 Edge 중 가중치의 값이 가장 작은 Edge을 선택한다.



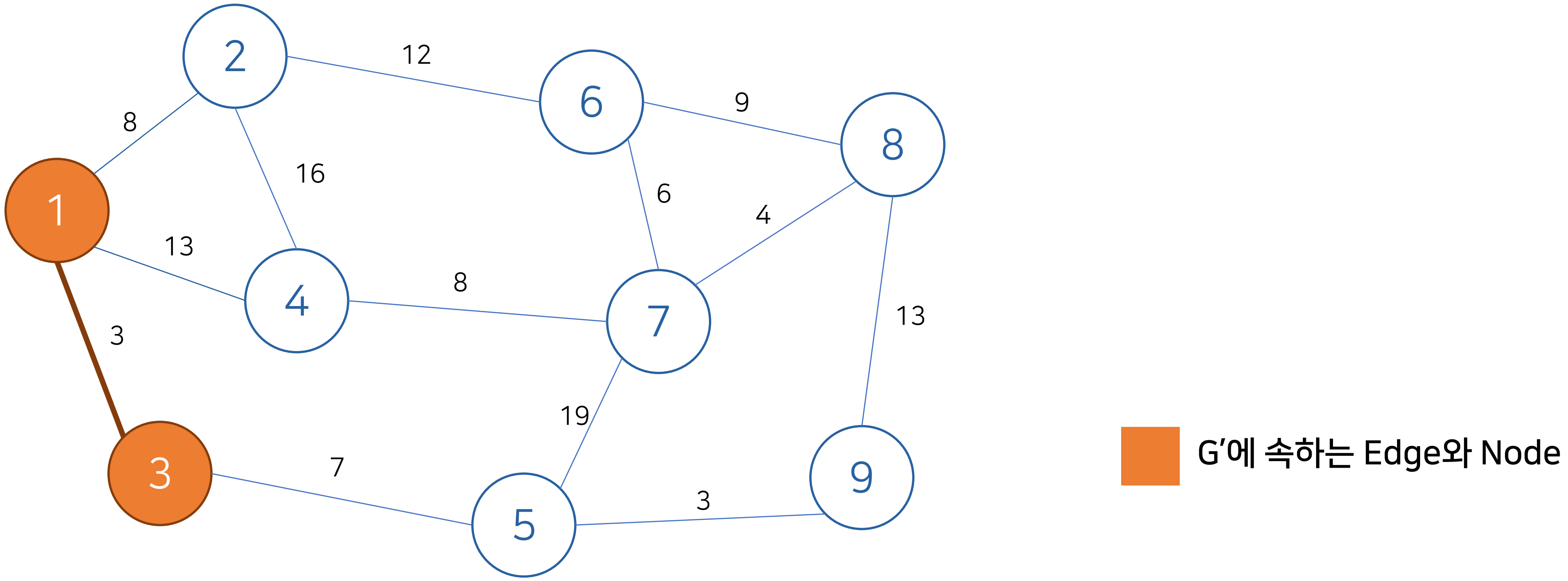
MCST – Prim's Algorithm

- G'에서 연결된 Edge를 선택하는데, 그 간선의 양쪽 Node가 G'에 모두 속하지 않는 Edge 중 가중치의 값이 가장 작은 Edge을 선택한다.



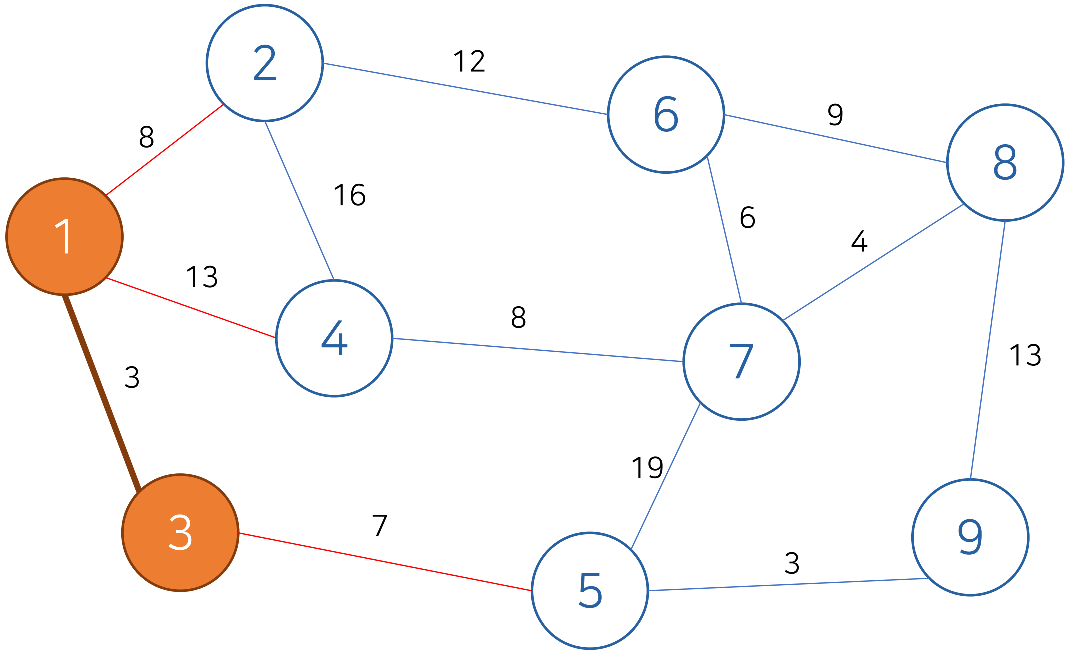
MCST – Prim’s Algorithm

- 그 Edge과 Edge에 연결된 Node를 G’에 추가한다.



MCST – Prim’s Algorithm

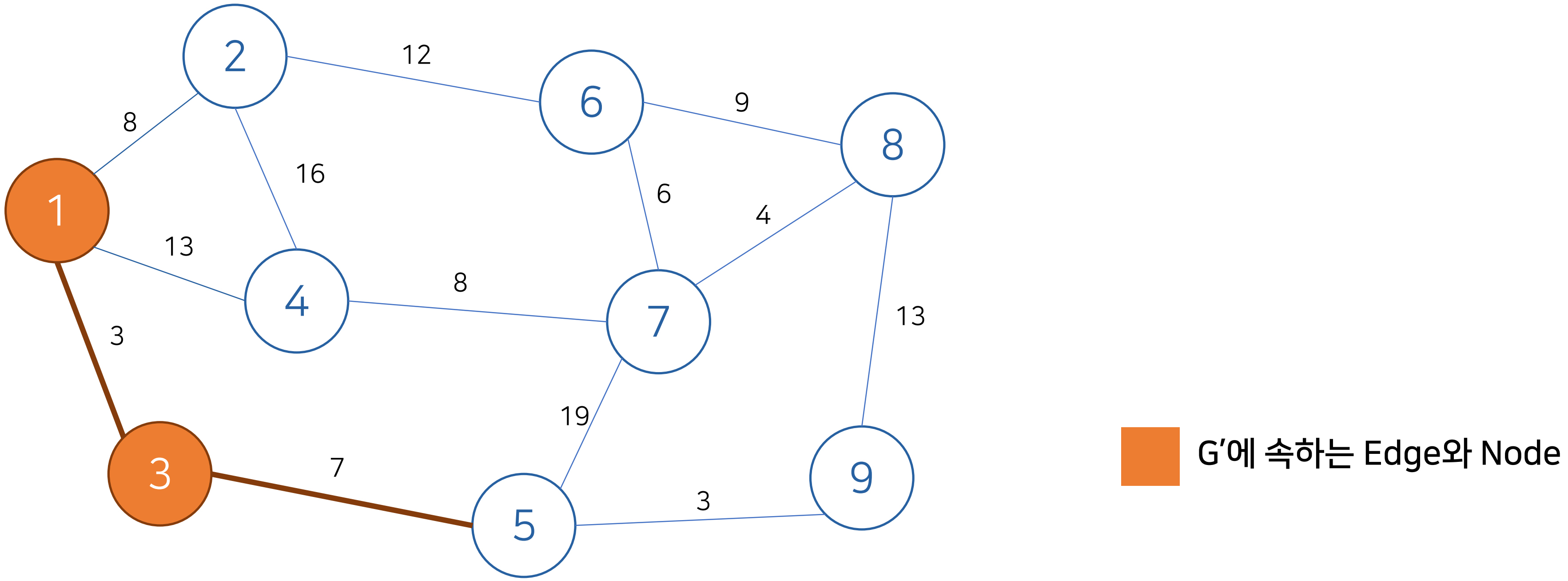
- G’에서 연결된 Edge를 선택하는데, 그 간선의 양쪽 Node가 G’에 모두 속하지 않는 Edge 중 가중치의 값이 가장 작은 Edge을 선택한다.



G’에 속하는 Edge와 Node

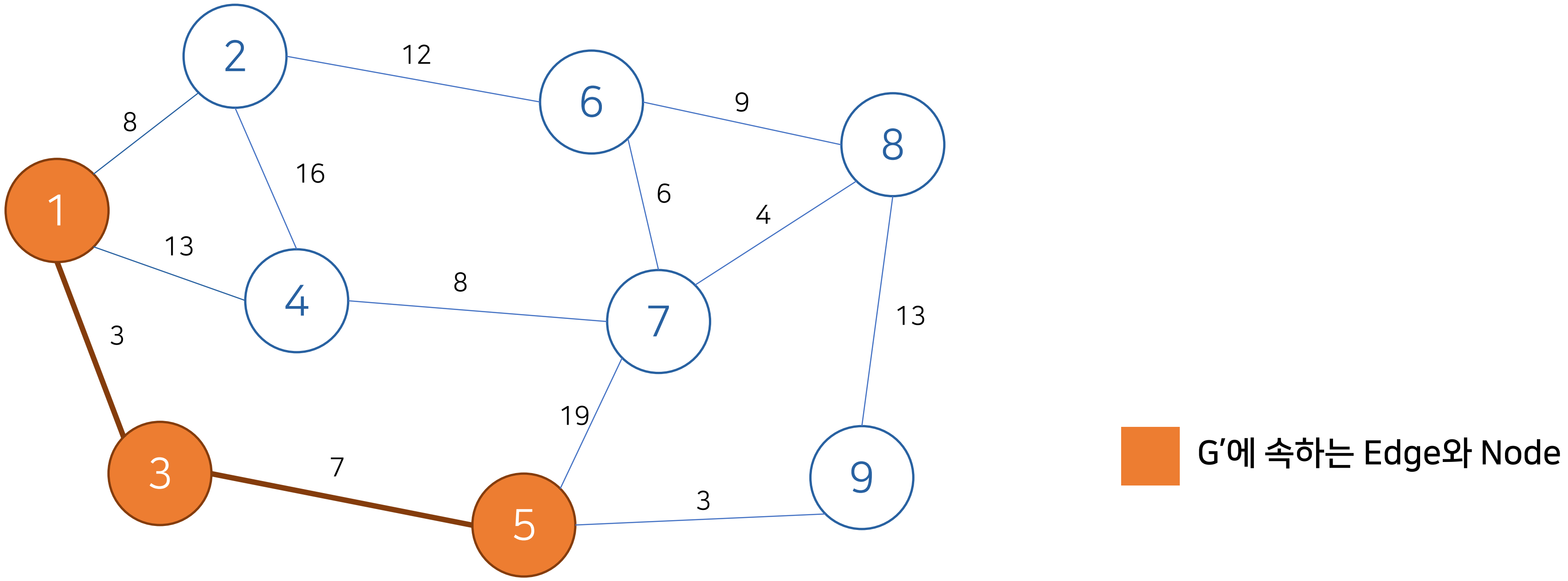
MCST – Prim’s Algorithm

- G’에서 연결된 Edge를 선택하는데, 그 간선의 양쪽 Node가 G’에 모두 속하지 않는 Edge 중 가중치의 값이 가장 작은 Edge을 선택한다.



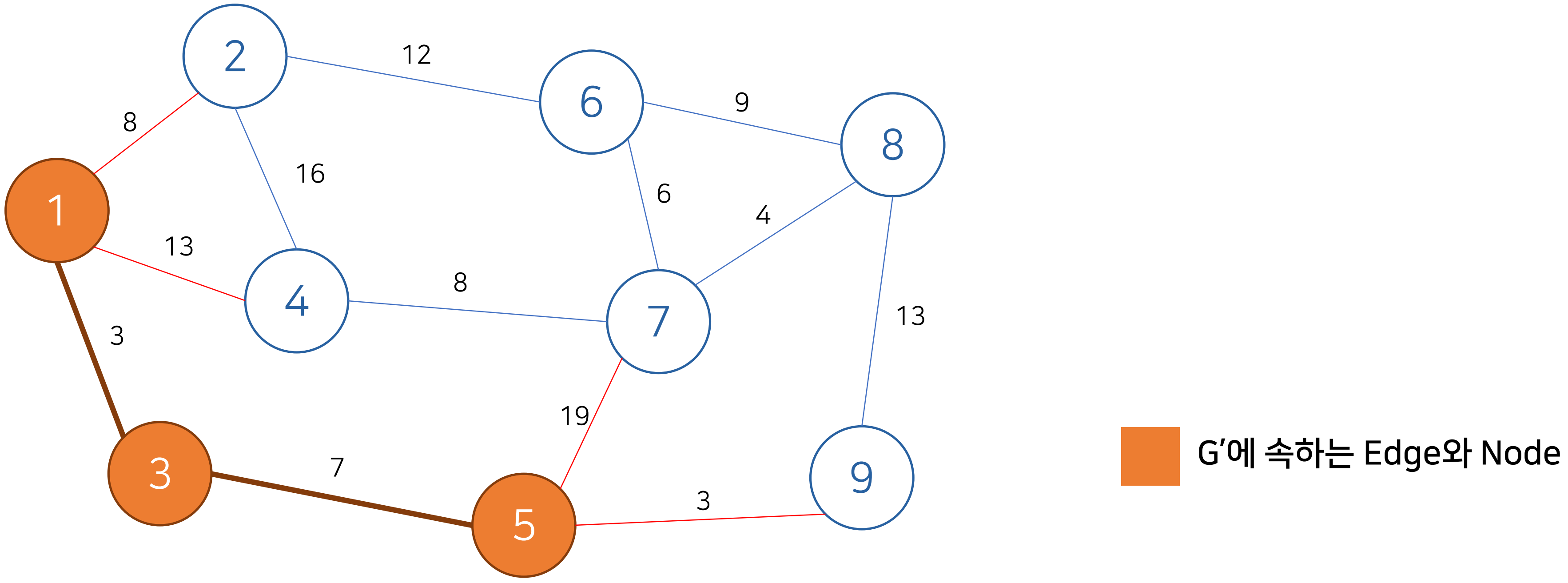
MCST – Prim's Algorithm

- 그 Edge과 Edge에 연결된 Node를 G'에 추가한다.



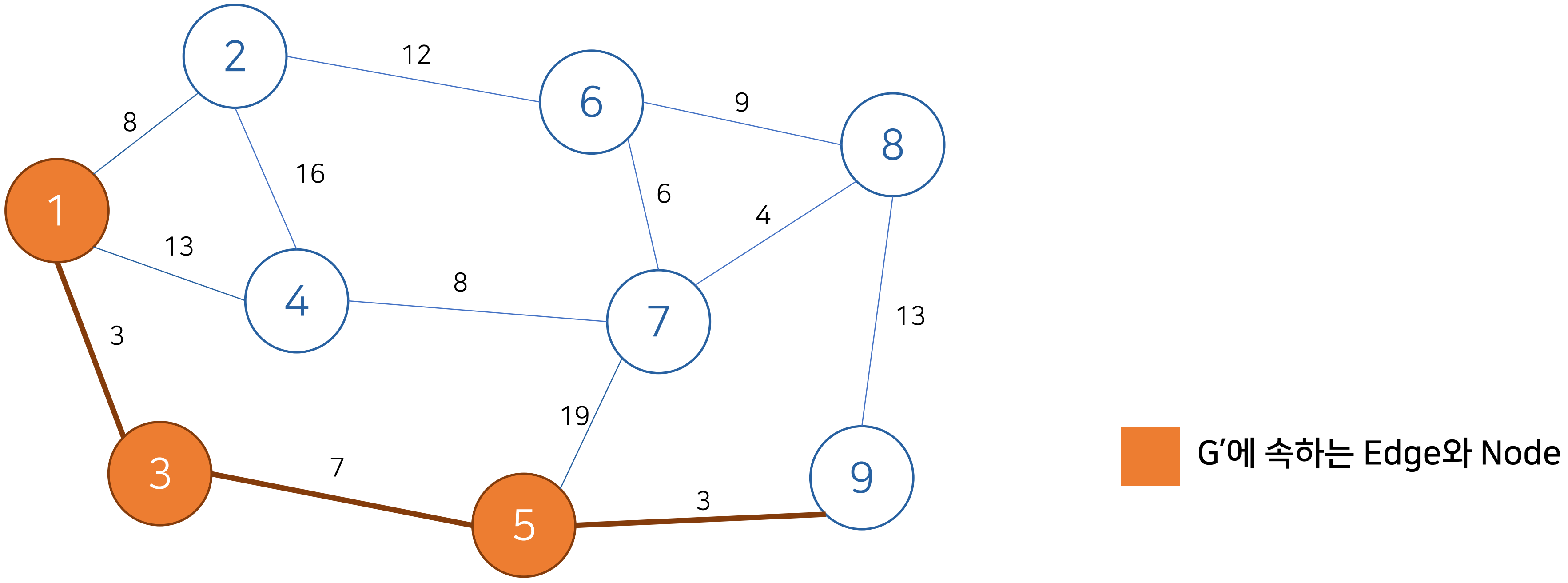
MCST – Prim's Algorithm

- G'에서 연결된 Edge를 선택하는데, 그 간선의 양쪽 Node가 G'에 모두 속하지 않는 Edge 중 가중치의 값이 가장 작은 Edge을 선택한다.



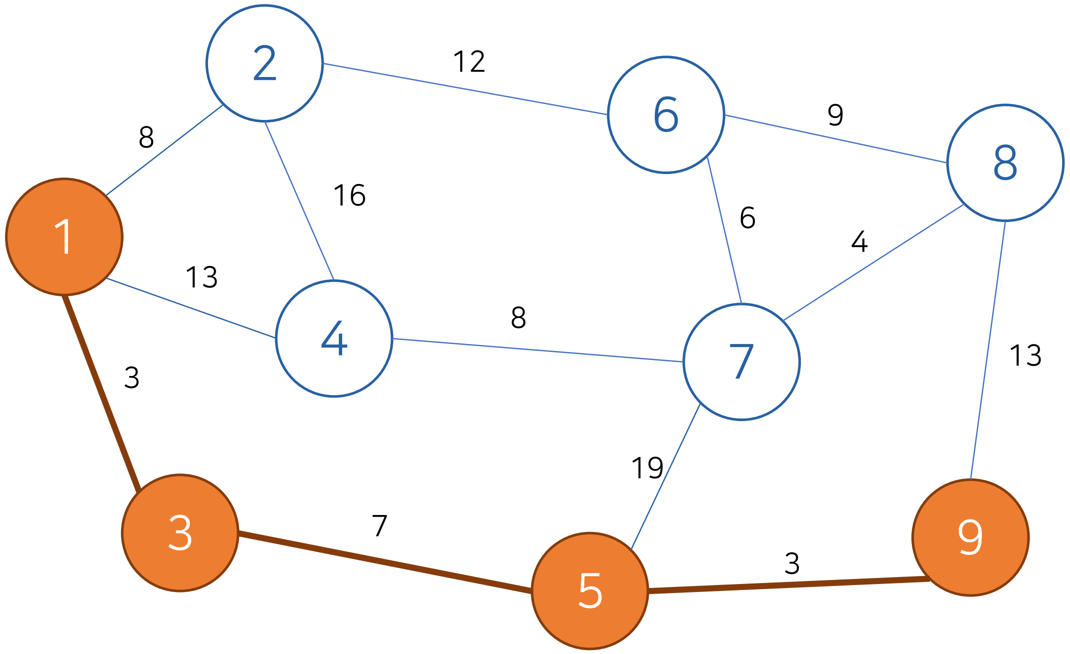
MCST – Prim’s Algorithm

- G’에서 연결된 Edge를 선택하는데, 그 간선의 양쪽 Node가 G’에 모두 속하지 않는 Edge 중 가중치의 값이 가장 작은 Edge을 선택한다.



MCST – Prim's Algorithm

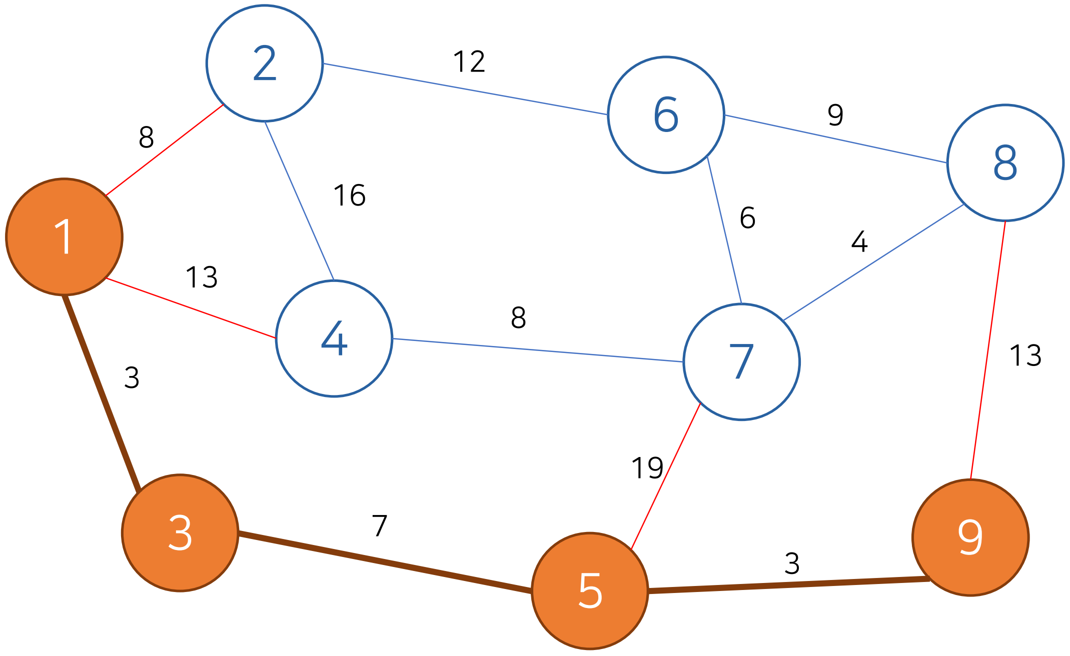
- 그 Edge과 Edge에 연결된 Node를 G'에 추가한다.



G'에 속하는 Edge와 Node

MCST – Prim's Algorithm

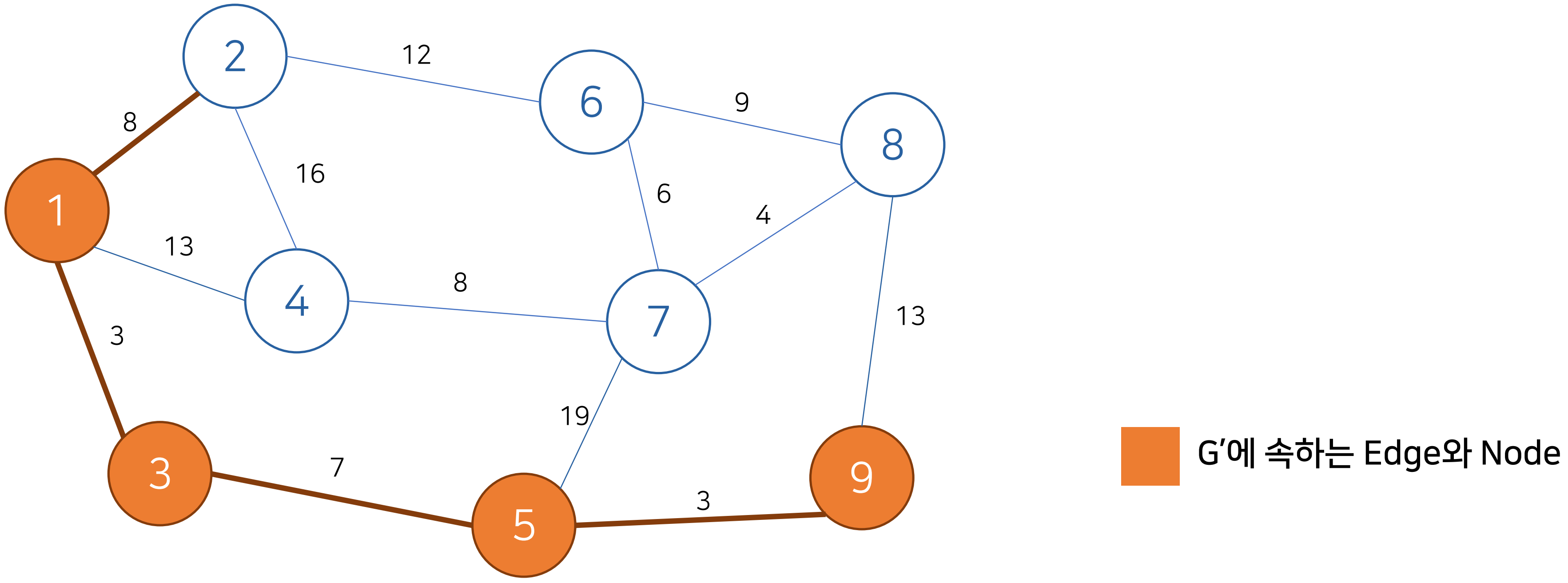
- G'에서 연결된 Edge를 선택하는데, 그 간선의 양쪽 Node가 G'에 모두 속하지 않는 Edge 중 가중치의 값이 가장 작은 Edge을 선택한다.



G'에 속하는 Edge와 Node

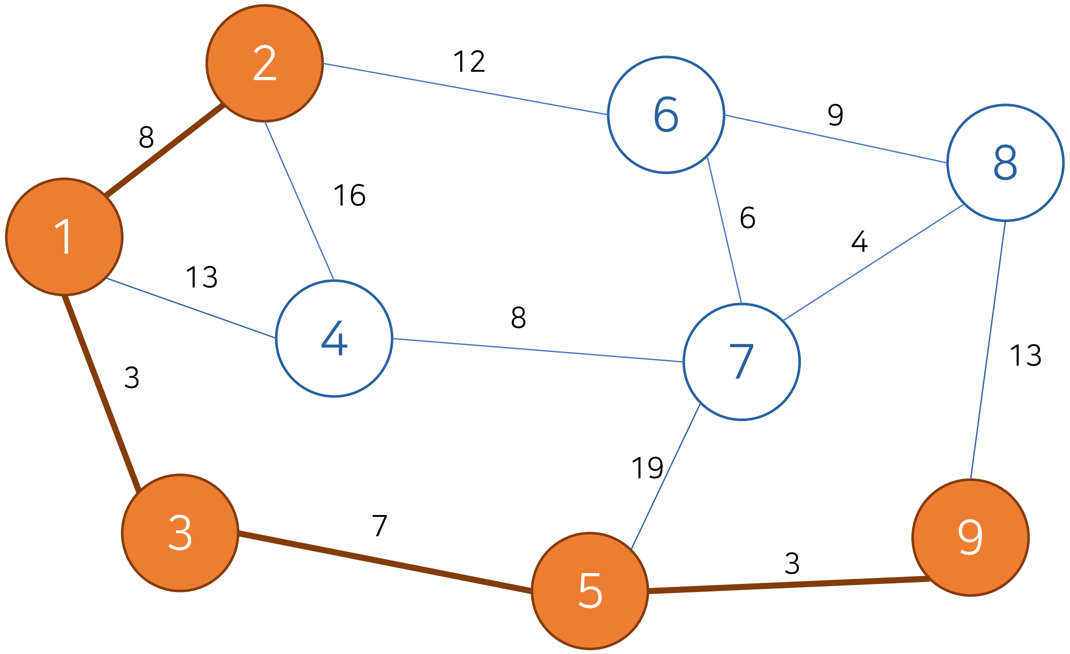
MCST – Prim's Algorithm

- G'에서 연결된 Edge를 선택하는데, 그 간선의 양쪽 Node가 G'에 모두 속하지 않는 Edge 중 가중치의 값이 가장 작은 Edge을 선택한다.



MCST – Prim's Algorithm

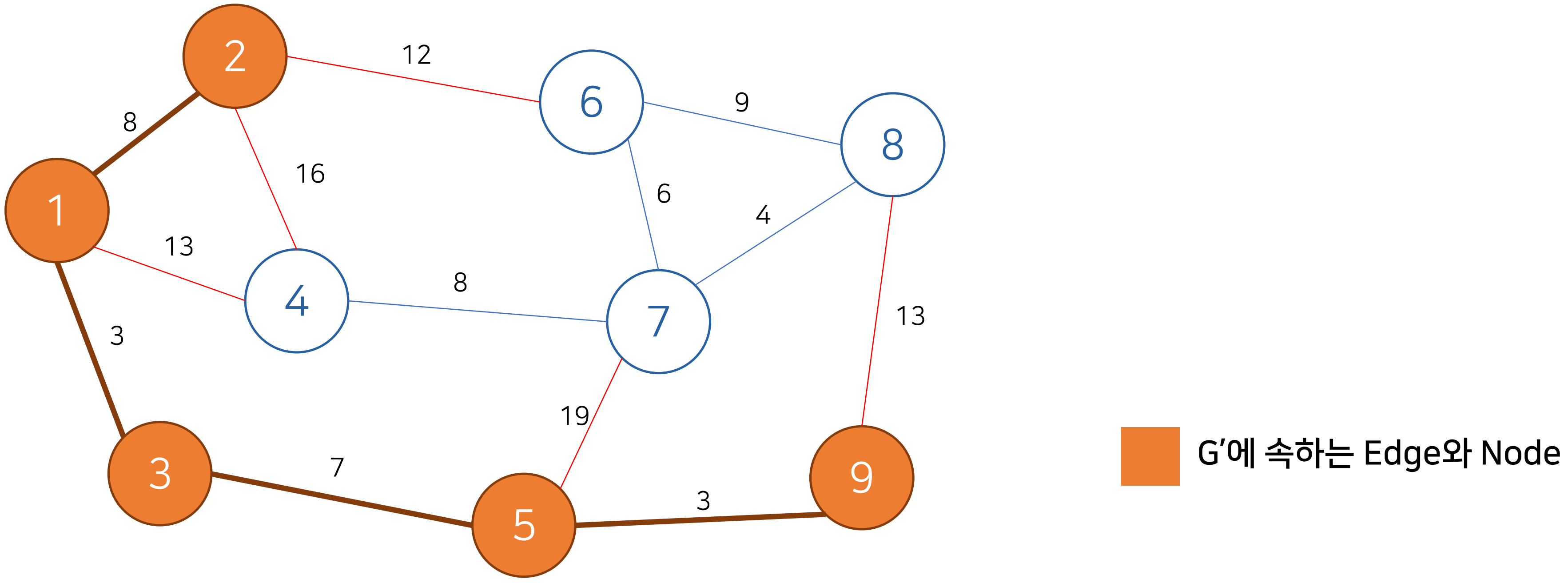
• 그 Edge과 Edge에 연결된 Node를 G'에 추가한다.



G'에 속하는 Edge와 Node

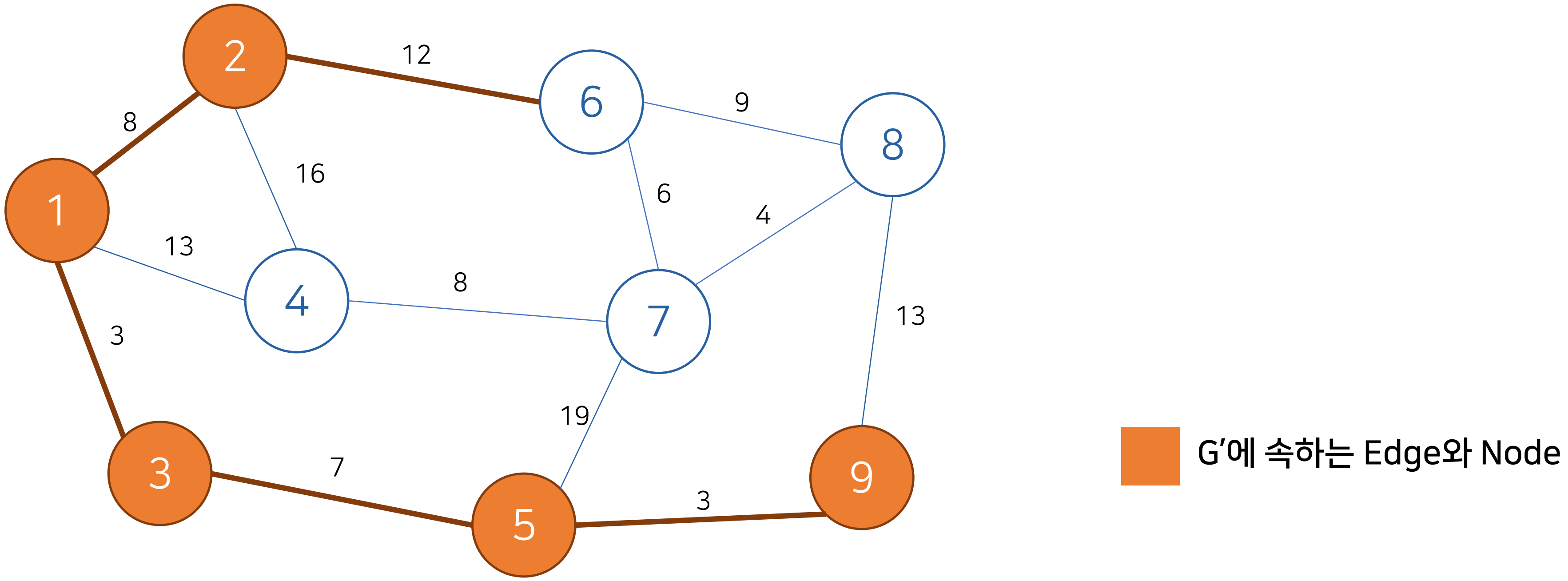
MCST – Prim's Algorithm

- G'에서 연결된 Edge를 선택하는데, 그 간선의 양쪽 Node가 G'에 모두 속하지 않는 Edge 중 가중치의 값이 가장 작은 Edge을 선택한다.



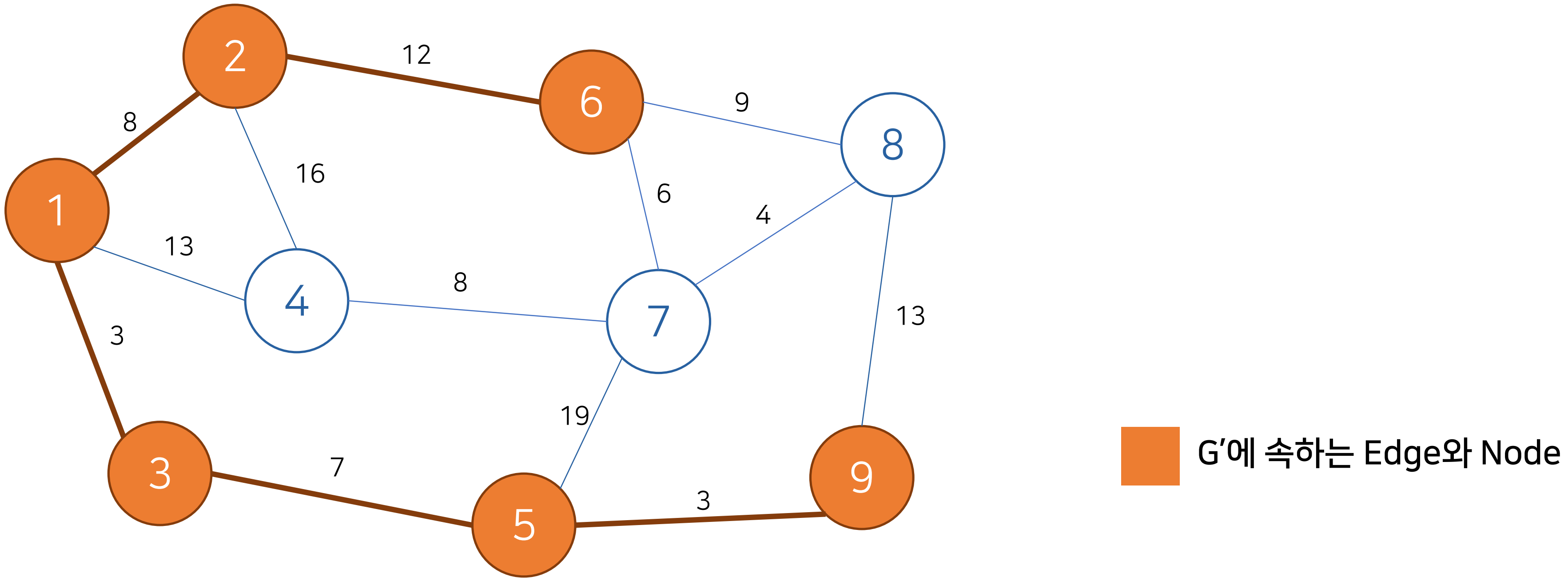
MCST – Prim's Algorithm

- G'에서 연결된 Edge를 선택하는데, 그 간선의 양쪽 Nod가e G'에 모두 속하지 않는 Edge 중 가중치의 값이 가장 작은 Edge을 선택한다.



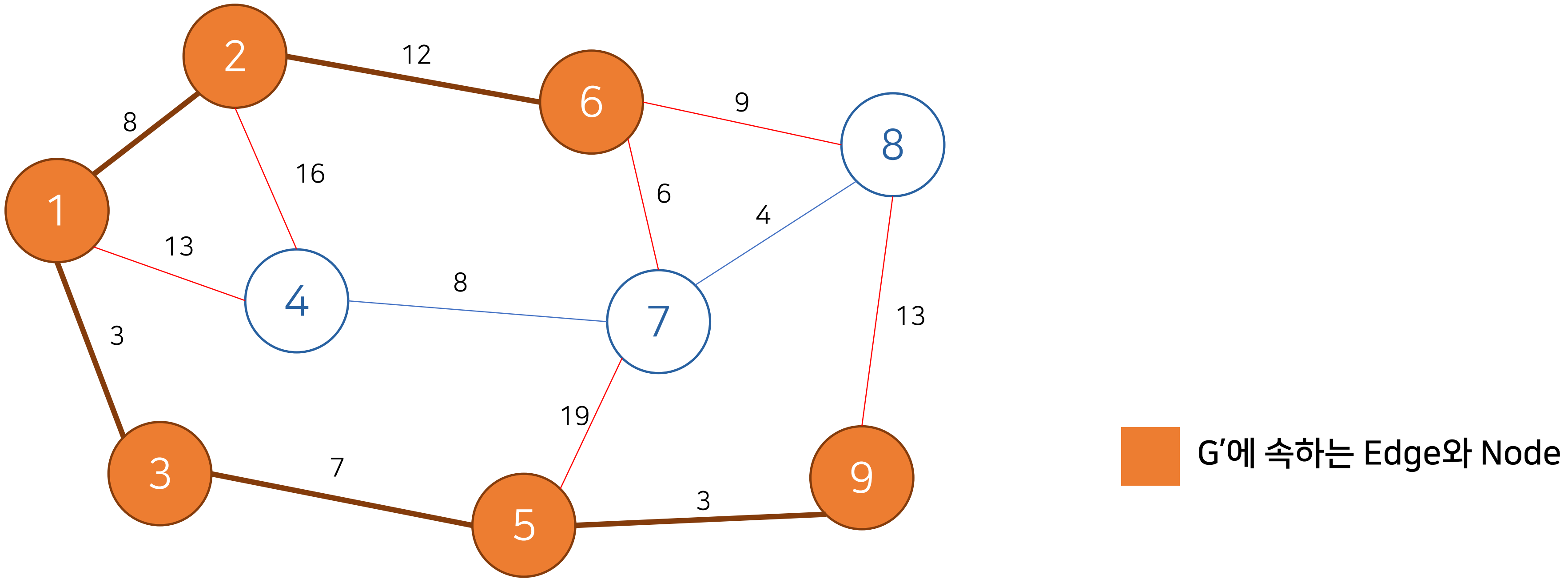
MCST – Prim's Algorithm

- 그 Edge과 Edge에 연결된 Node를 G'에 추가한다.



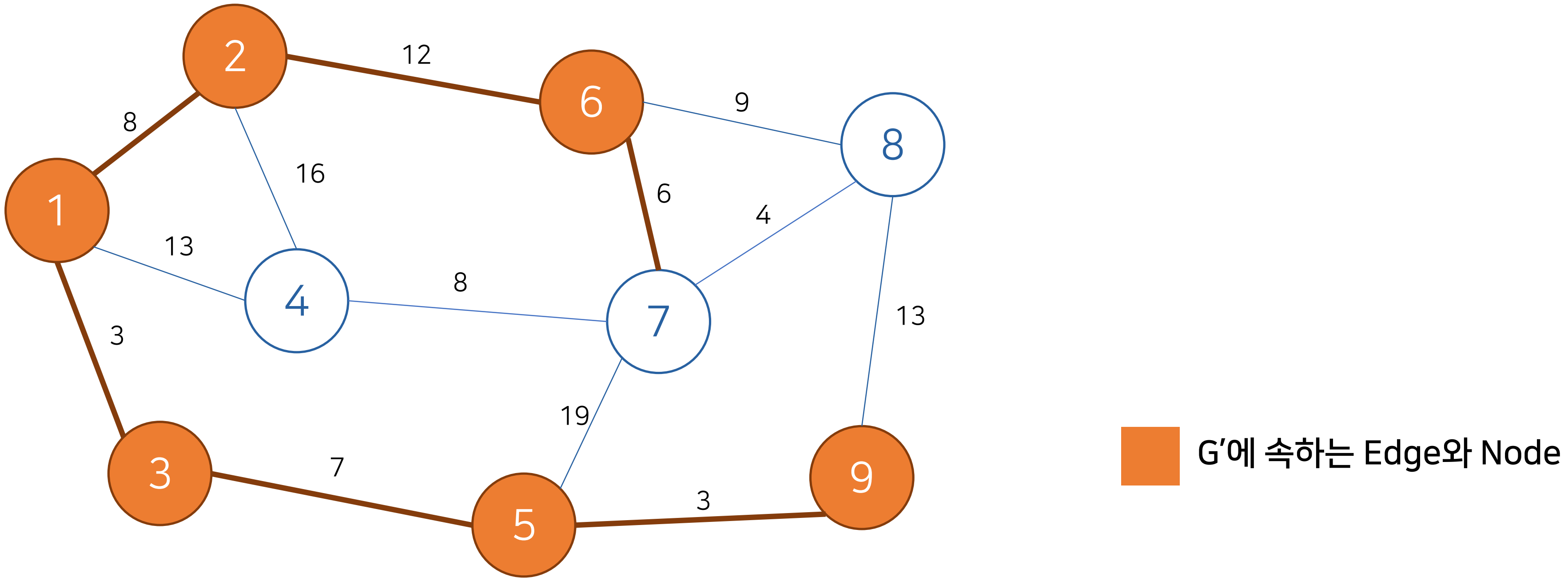
MCST – Prim's Algorithm

- G'에서 연결된 Edge를 선택하는데, 그 간선의 양쪽 Node가 G'에 모두 속하지 않는 Edge 중 가중치의 값이 가장 작은 Edge을 선택한다.



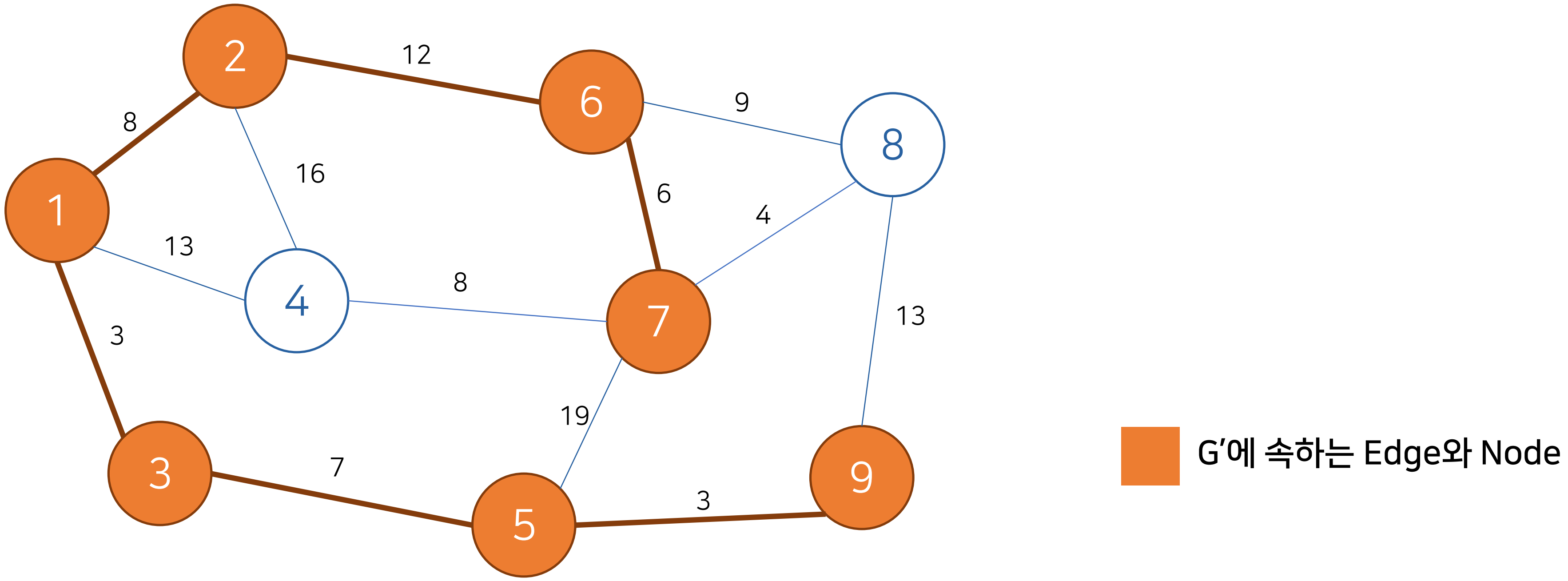
MCST – Prim's Algorithm

- G'에서 연결된 Edge를 선택하는데, 그 간선의 양쪽 Node가 G'에 모두 속하지 않는 Edge 중 가중치의 값이 가장 작은 Edge을 선택한다.



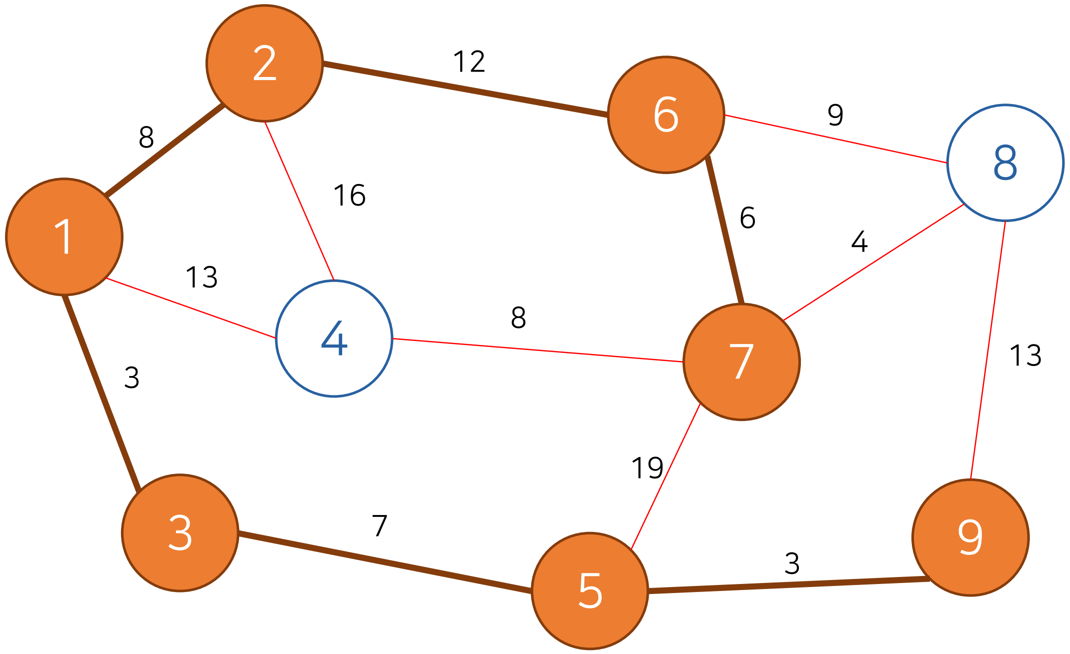
MCST – Prim's Algorithm

- 그 Edge과 Edge에 연결된 Node를 G'에 추가한다.



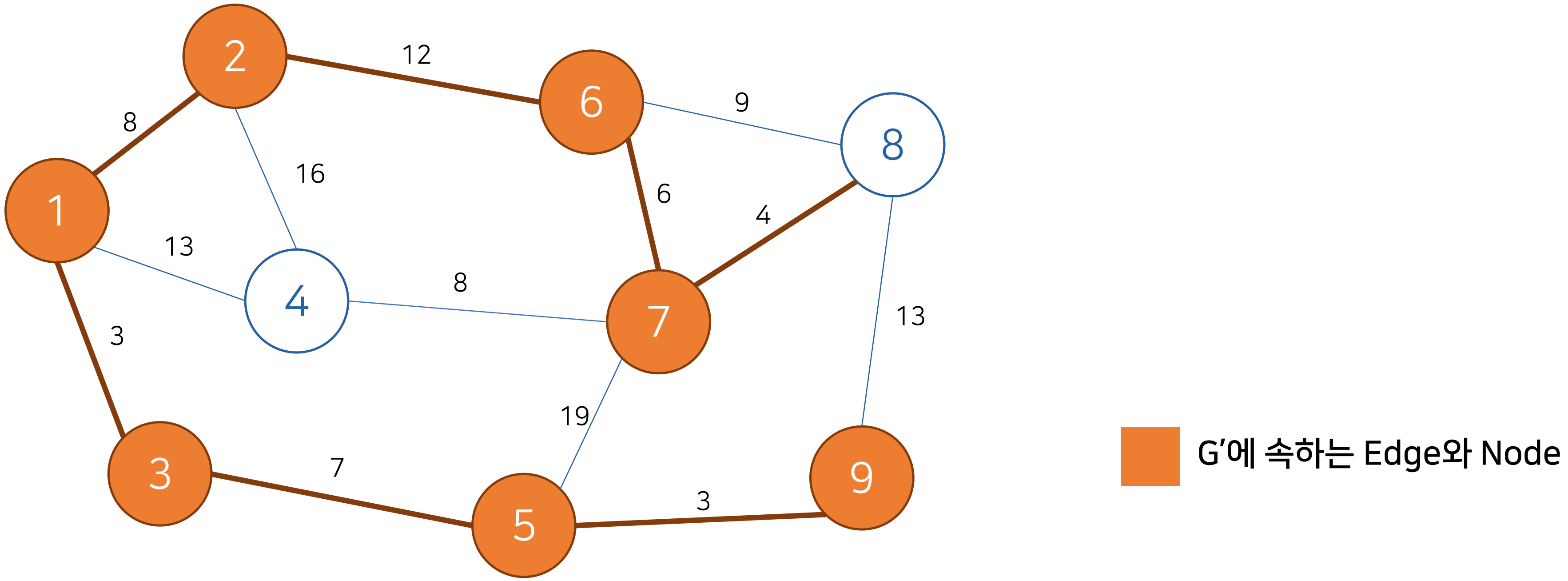
MCST – Prim's Algorithm

- G'에서 연결된 Edge를 선택하는데, 그 간선의 양쪽 Node가 G'에 모두 속하지 않는 Edge 중 가중치의 값이 가장 작은 Edge을 선택한다.



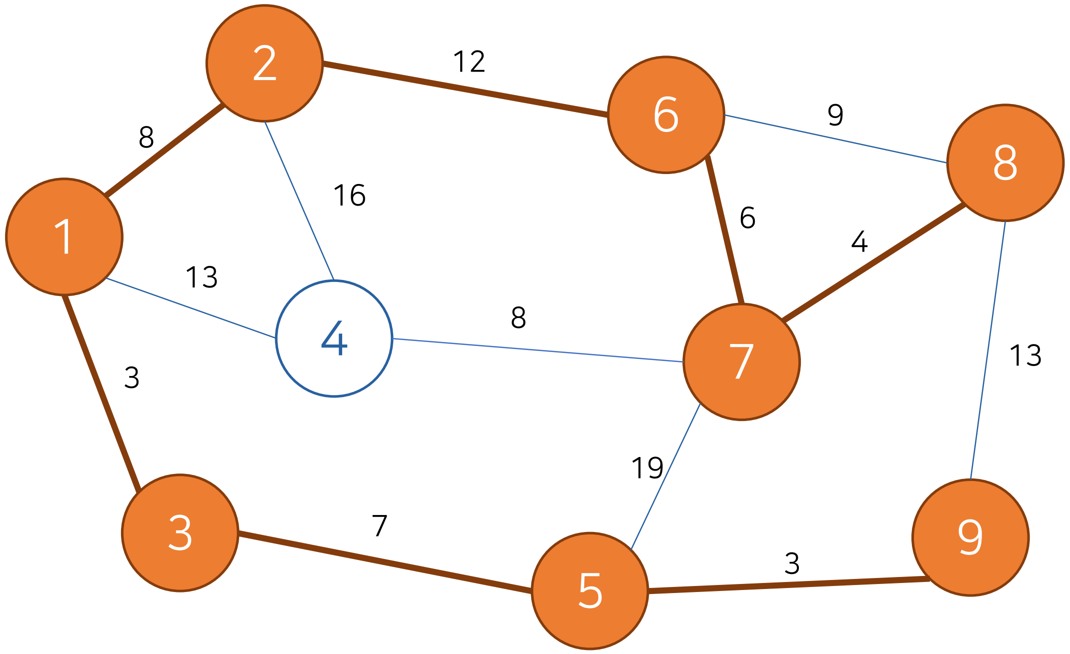
MCST – Prim's Algorithm

- G'에서 연결된 Edge를 선택하는데, 그 간선의 양쪽 Node가 G'에 모두 속하지 않는 Edge 중 가중치의 값이 가장 작은 Edge을 선택한다.



MCST – Prim's Algorithm

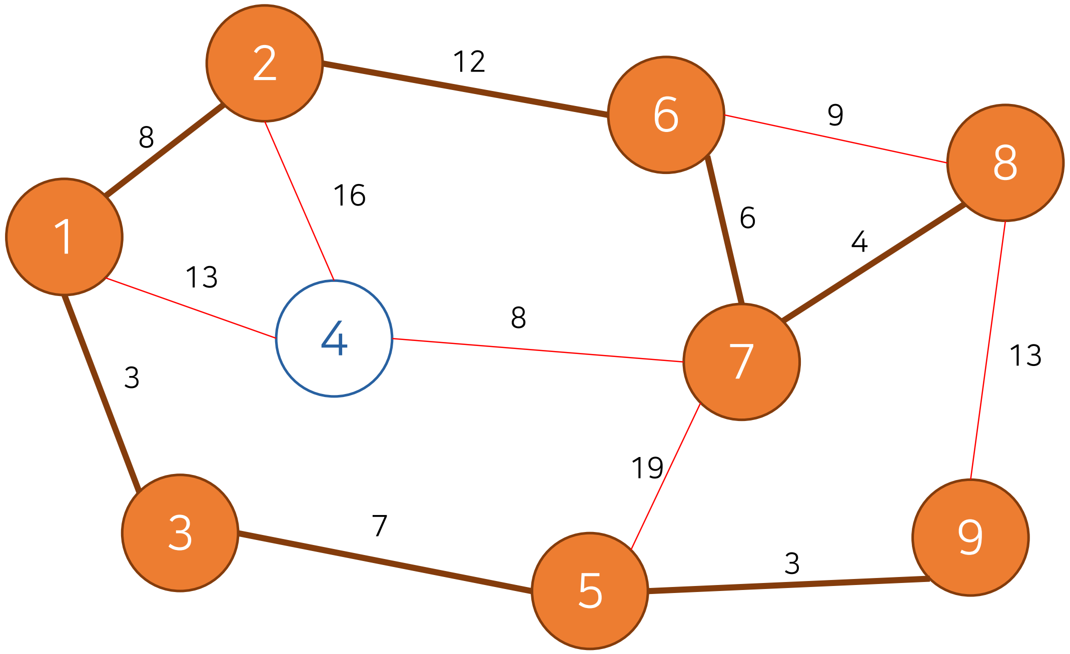
- 그 Edge과 Edge에 연결된 Node를 G'에 추가한다.



G'에 속하는 Edge와 Node

MCST – Prim’s Algorithm

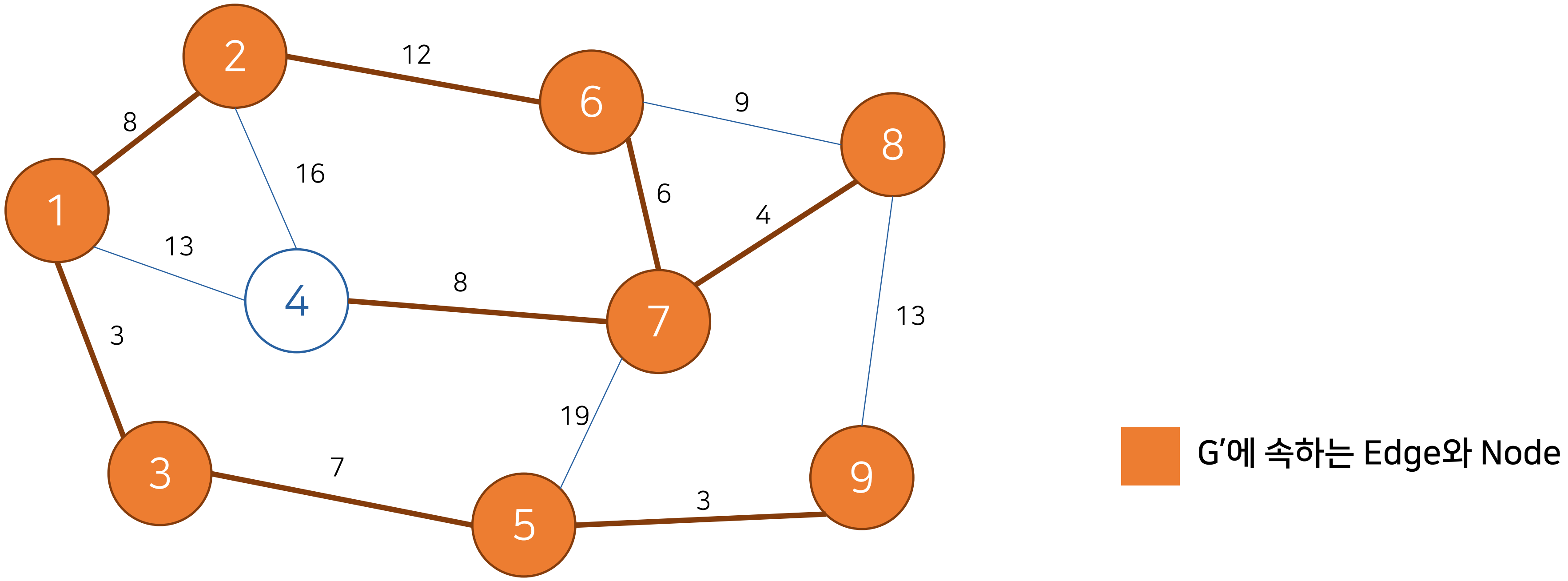
- G’에서 연결된 Edge를 선택하는데, 그 간선의 양쪽 Node가 G’에 모두 속하지 않는 Edge 중 가중치의 값이 가장 작은 Edge을 선택한다.



G’에 속하는 Edge와 Node

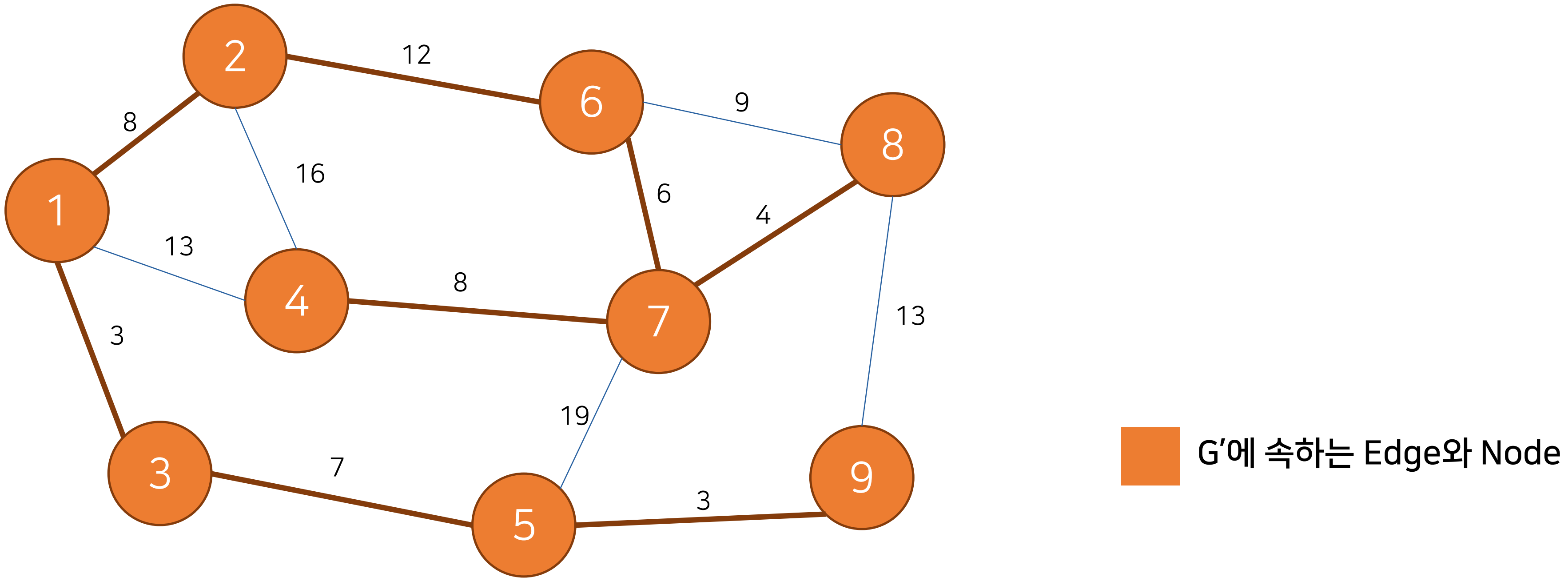
MCST – Prim's Algorithm

- G'에서 연결된 Edge를 선택하는데, 그 간선의 양쪽 Node가 G'에 모두 속하지 않는 Edge 중 가중치의 값이 가장 작은 Edge을 선택한다.



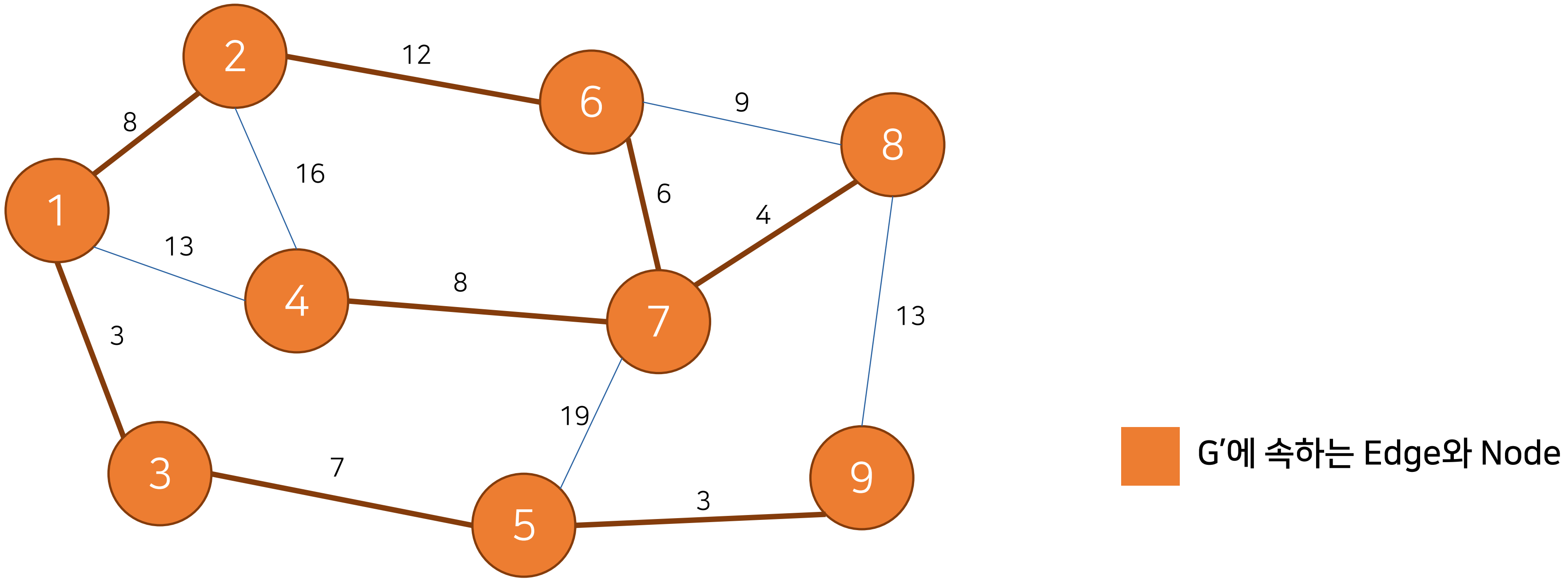
MCST – Prim's Algorithm

- 그 Edge과 Edge에 연결된 Node를 G'에 추가한다.



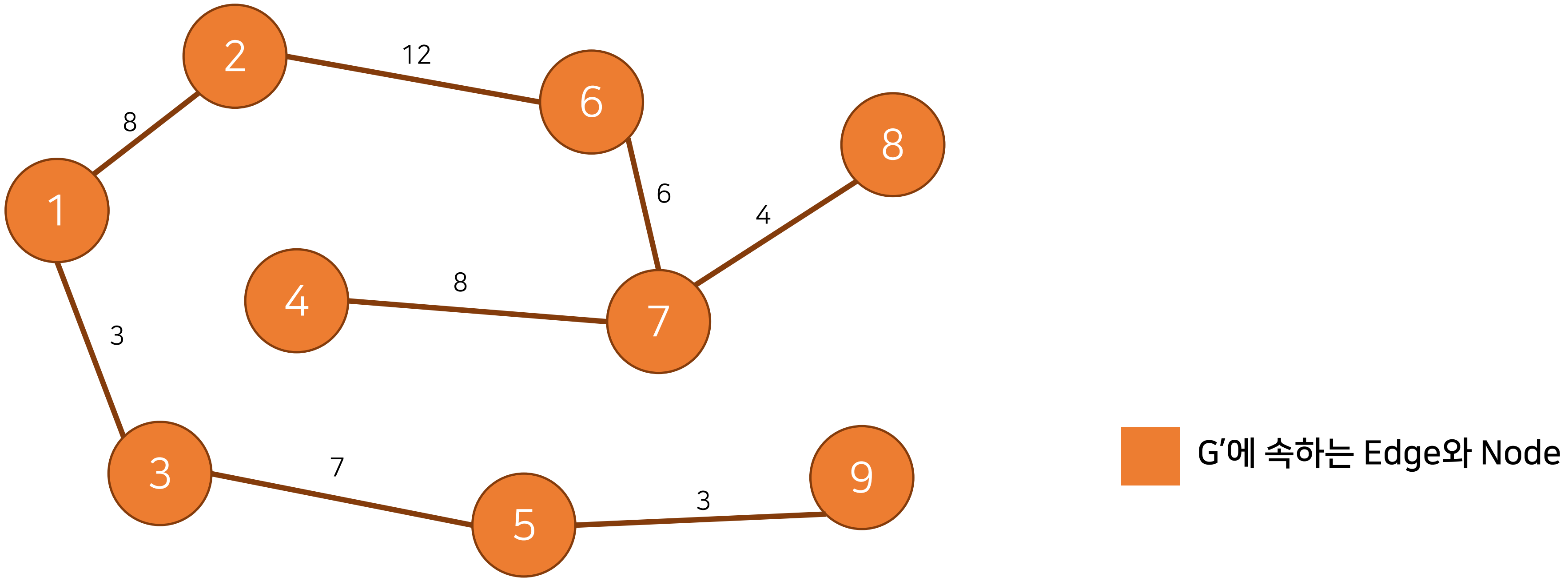
MCST – Prim's Algorithm

- 모든 Node들이 G'에 포함되어 있으므로 종료한다.



MCST – Prim's Algorithm

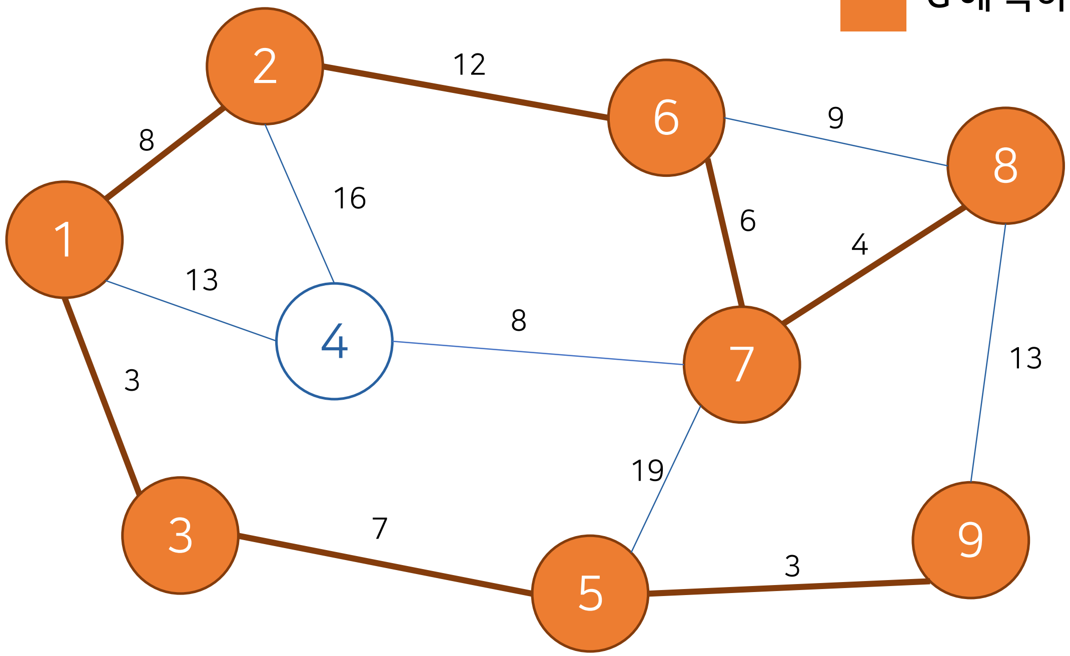
- 이때 G'가 Minimum Cost Spanning Tree이다.

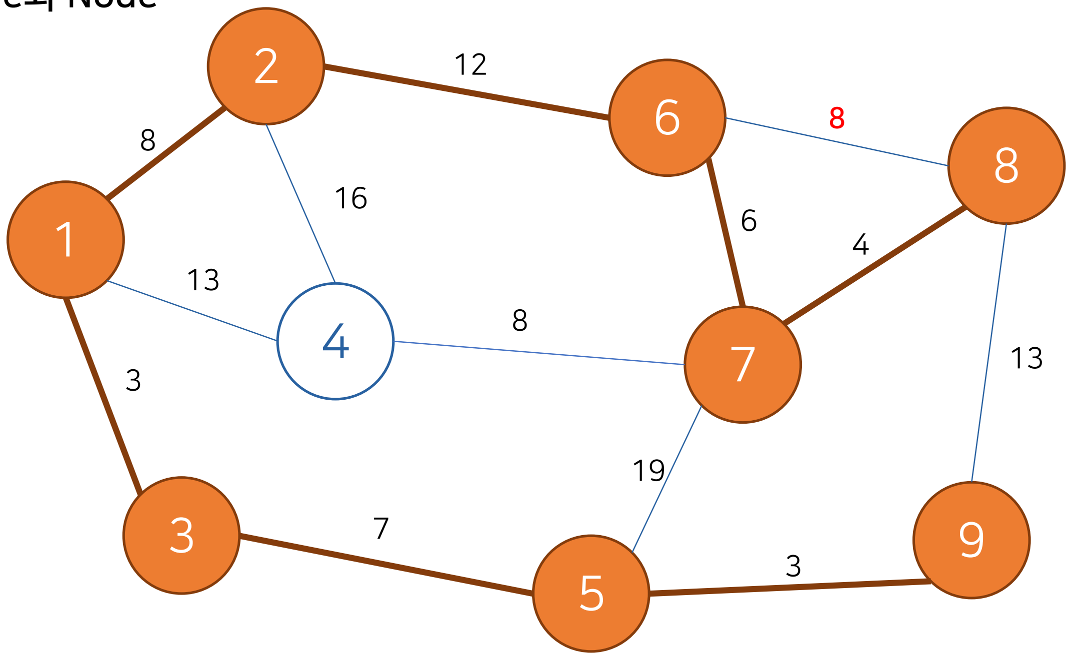


MCST – Prim's Algorithm

- 만약 Edge 6-8의 가중치가 8인 다음 상황에서 어떤 Edge를 선택해야 할까?

G'에 속하는 Edge와 Node

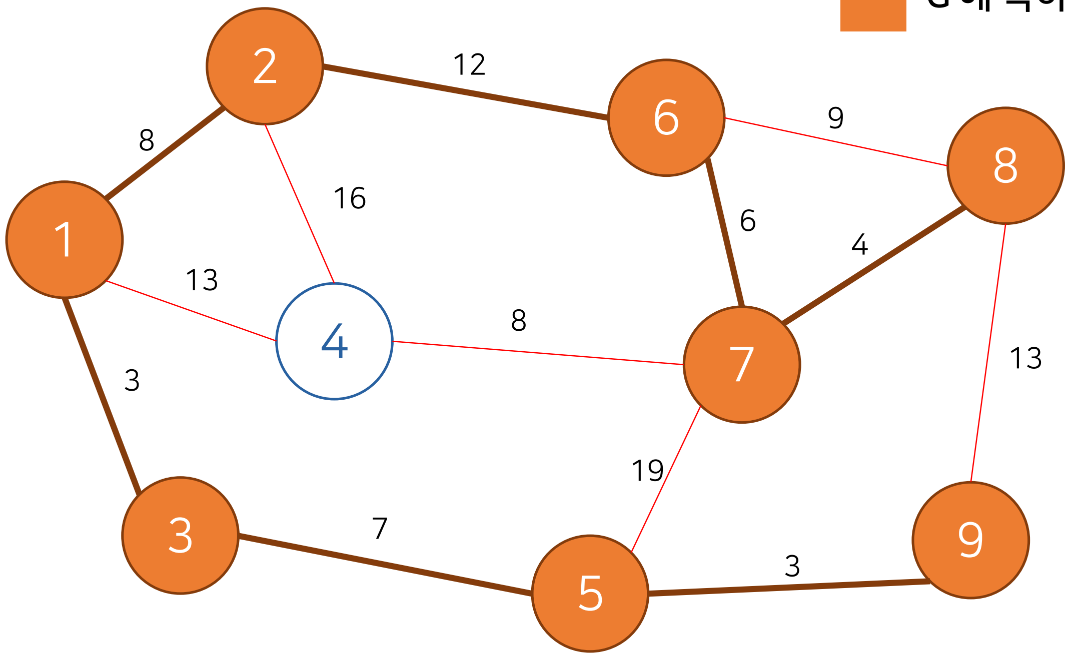


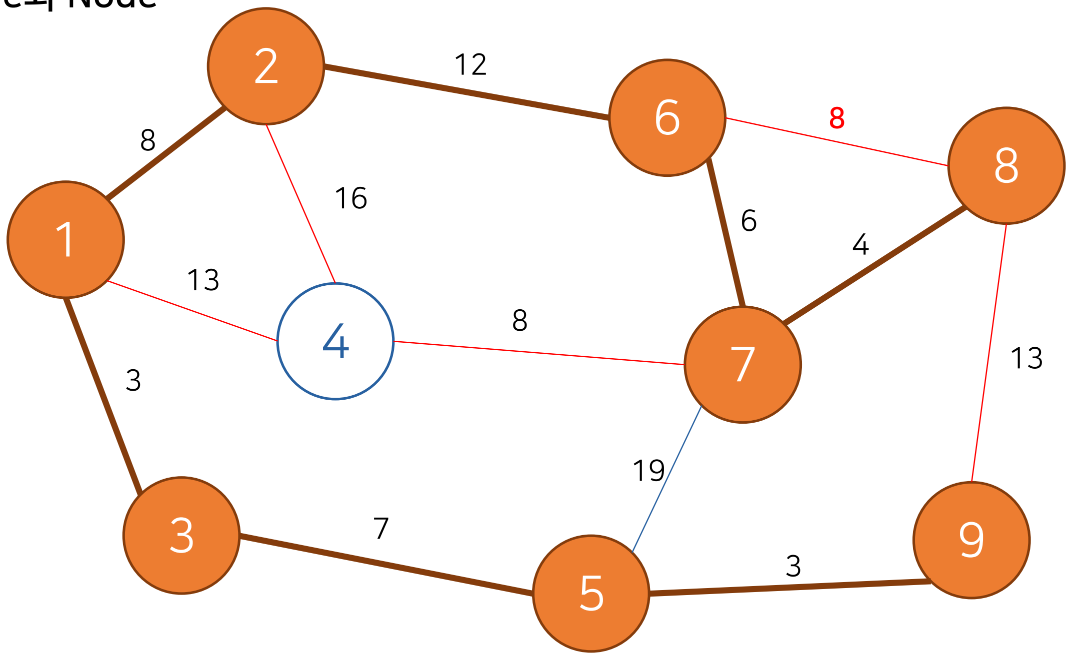


MCST – Prim's Algorithm

- 만약 Edge 6-8의 가중치가 8인 다음 상황에서 어떤 Edge를 선택해야 할까?

G'에 속하는 Edge와 Node

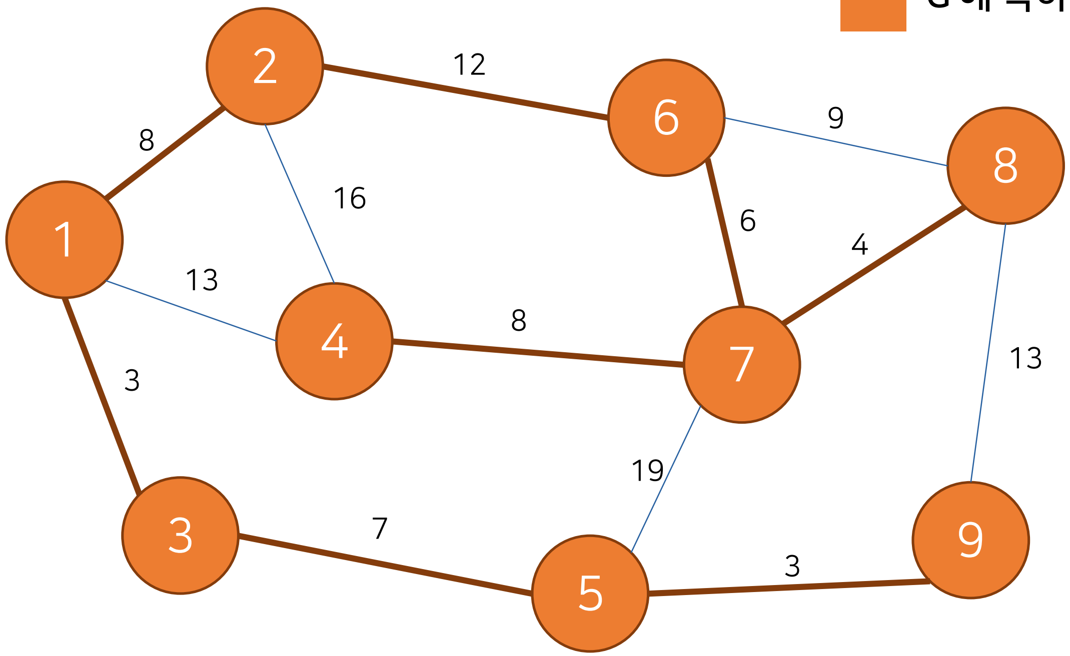


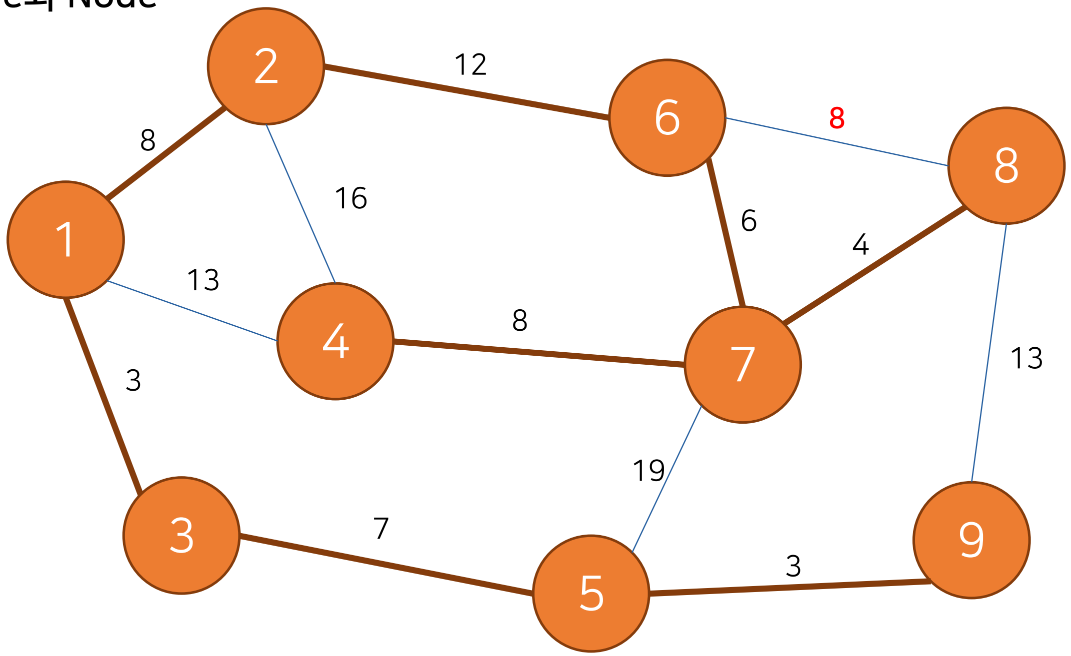


MCST – Prim's Algorithm

- 만약 Edge 6-8의 가중치가 8인 다음 상황에서 어떤 Edge를 선택해야 할까?

G'에 속하는 Edge와 Node

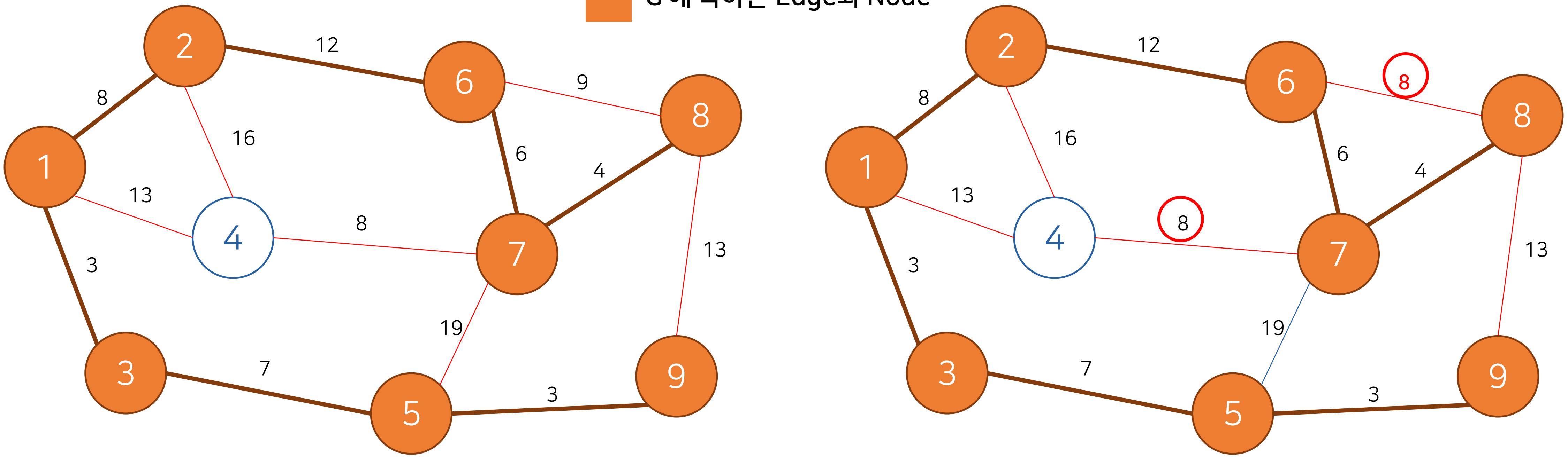




MCST – Prim's Algorithm

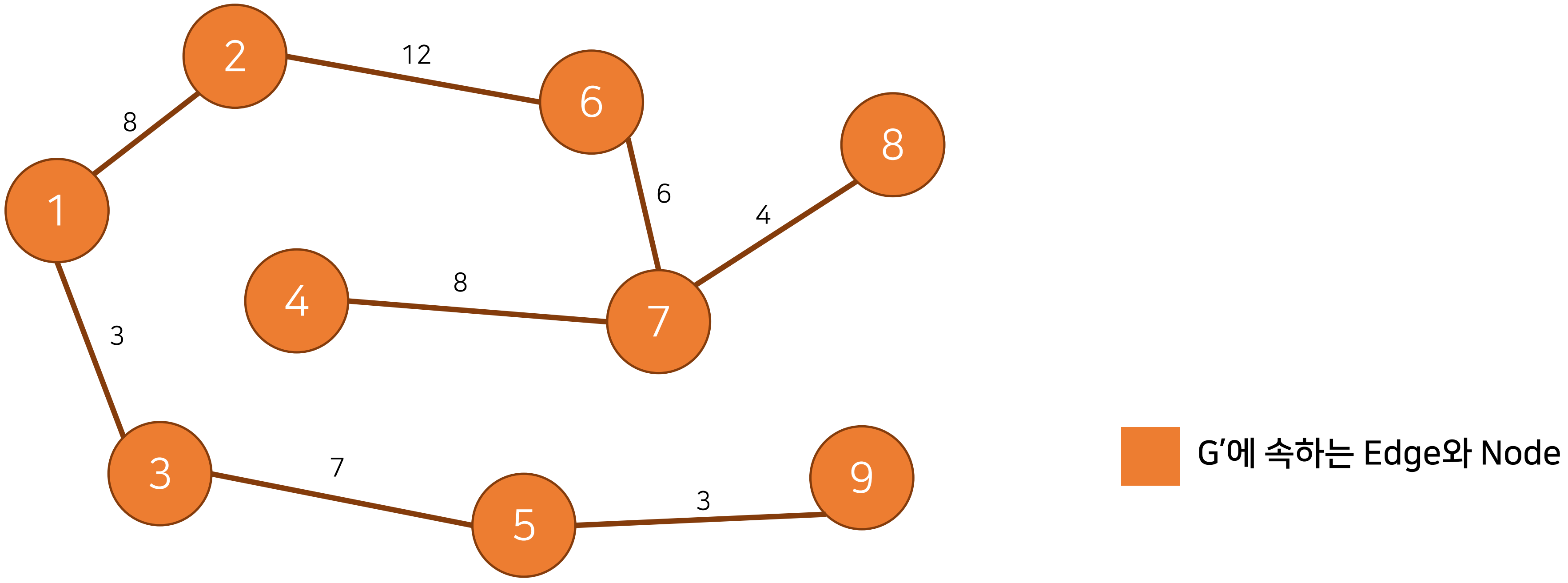
- G'에서 연결된 Edge를 선택하는데, 그 간선의 양쪽 Node가 G'에 모두 속하지 않는 Edge 중 가중치의 값이 가장 작은 Edge을 선택한다.

G'에 속하는 Edge와 Node



MCST – Prim's Algorithm

- Minimum Cost = 51



MCST – Prim's Algorithm

- Time Complexity
  - Adjacency List : $O(V^2)$
  - Binary Heap : $O(E\log V)$

MCST

BOJ 1197 최소 스패닝 트리
BOJ 1922 네트워크 연결
BOJ 9372 상근이의 여행
BOJ 2887 행성 터널

수고하셨습니다!