MST

신장 트리(spanning tree)

신장 트리
(그래프 G의 부분 그래프 G')
1. 연결되어 있다(Connected)
2. 최소 에지 수
|E| = |V| - 1 → 트리의 조건
3. E(G') ⊆ E(G)
4. V(G') = V(G)

최소 비용 신장 트리(minimum cost spanning tree)

최소 비용 신장 트리 T
1. 무방향 그래프
2. 가중치 그래프
3. 비용(cost)이 최소가 된다.

$$\sum_{e \in E(T)} w[e]$$

최소 비용 신장 트리(MST) 알고리즘

탐욕 알고리즘(Greedy algorithm)

1. **Kruskal algorithm**
2. **Prim algorithm**
3. **Sollin algorithm**

탐욕 알고리즘(Greedy algorithm)

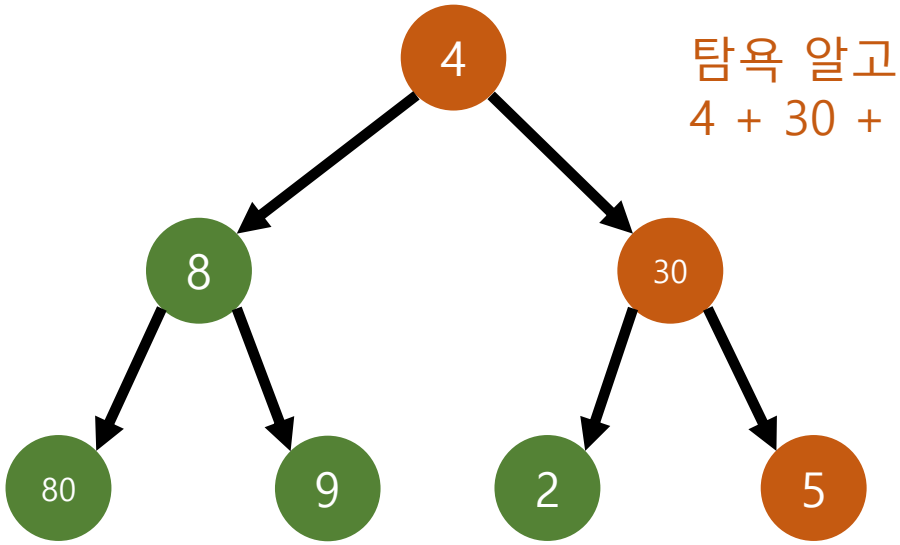
탐욕 알고리즘
: 전역적 최적해(global optimum)를 찾기 위해
각 스테이지마다 지역적 최적해(local optimum)를 선택

지역적 최적 선택의 모음이므로
전역적으로 최적해라는 보장이 없다!!

탐욕 알고리즘(Greedy algorithm)

탐욕 알고리즘이 실패하는 예

최대 합을 구하는 문제



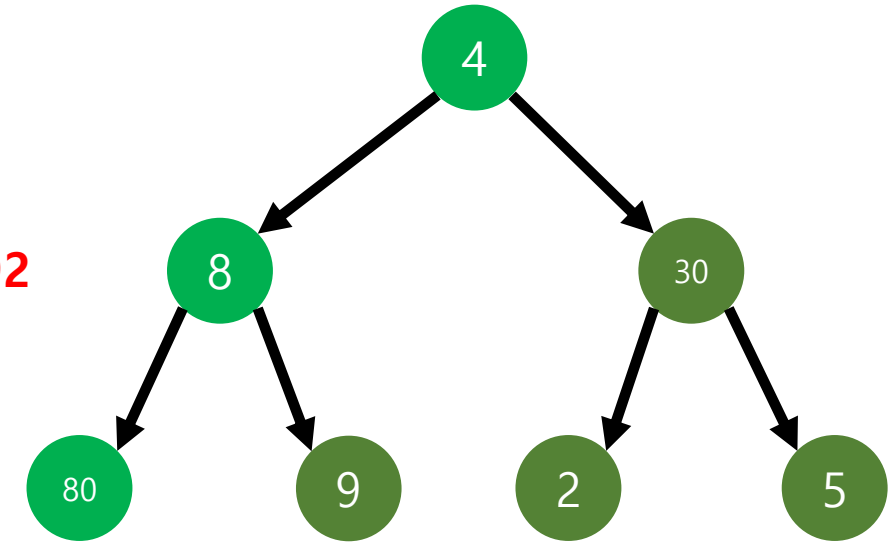
탐욕 알고리즘에 의한 경로
4 + 30 + 5 = 39

탐욕 알고리즘(Greedy algorithm)

탐욕 알고리즘이 실패하는 예

최대 합을 구하는 문제

실제 최적 경로
4 + 8 + 80 = 92



탐욕 알고리즘(Greedy algorithm)

탐욕 알고리즘이 성공하기 위한 조건

1. Greedy choice property
   : 지역 최적해를 선택해 나가면
   전역 최적해에 도달할 수 있다
2. Optimal substructure
   : 문제에 대한 최적해가
   부분 문제에 대한 최적해를 포함할 때

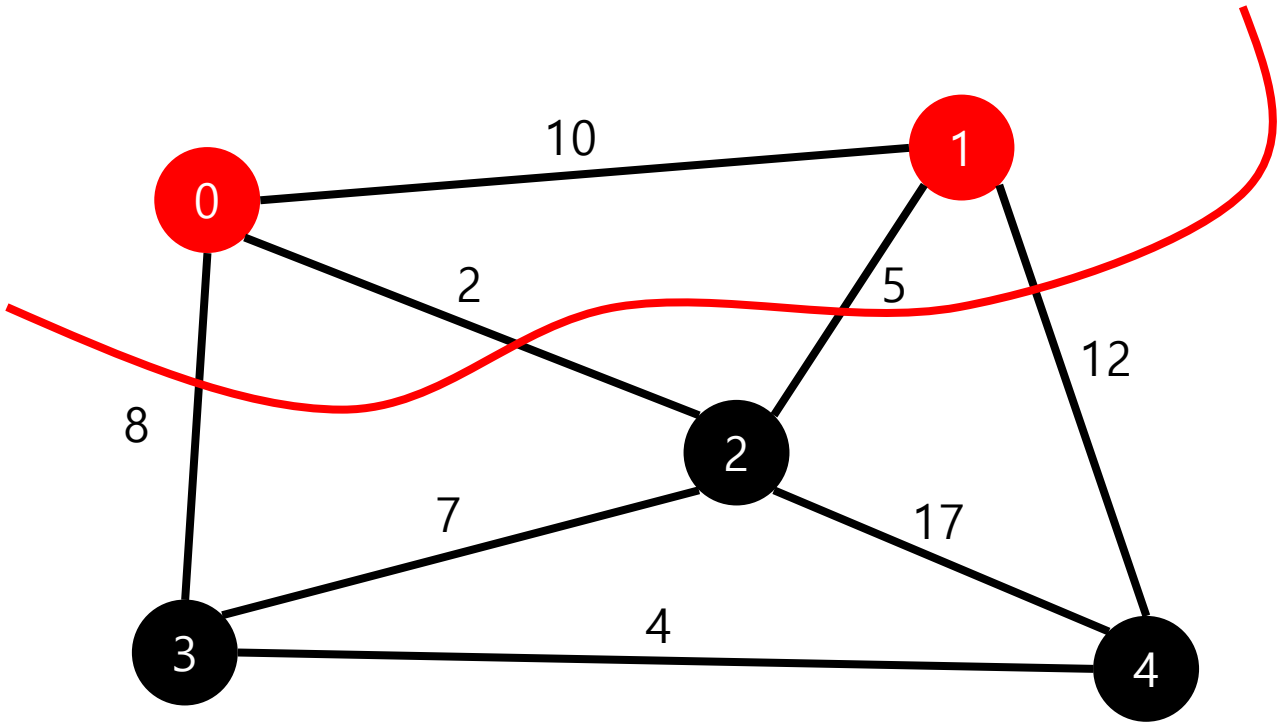
Greedy MST algorithm

단순화 가정
- 에지 가중치는 서로 다르다(distinct)
- 그래프는 연결되어 있다(connected)

결과
- 최소 비용 신장 트리가 존재하며 유일하다(unique)

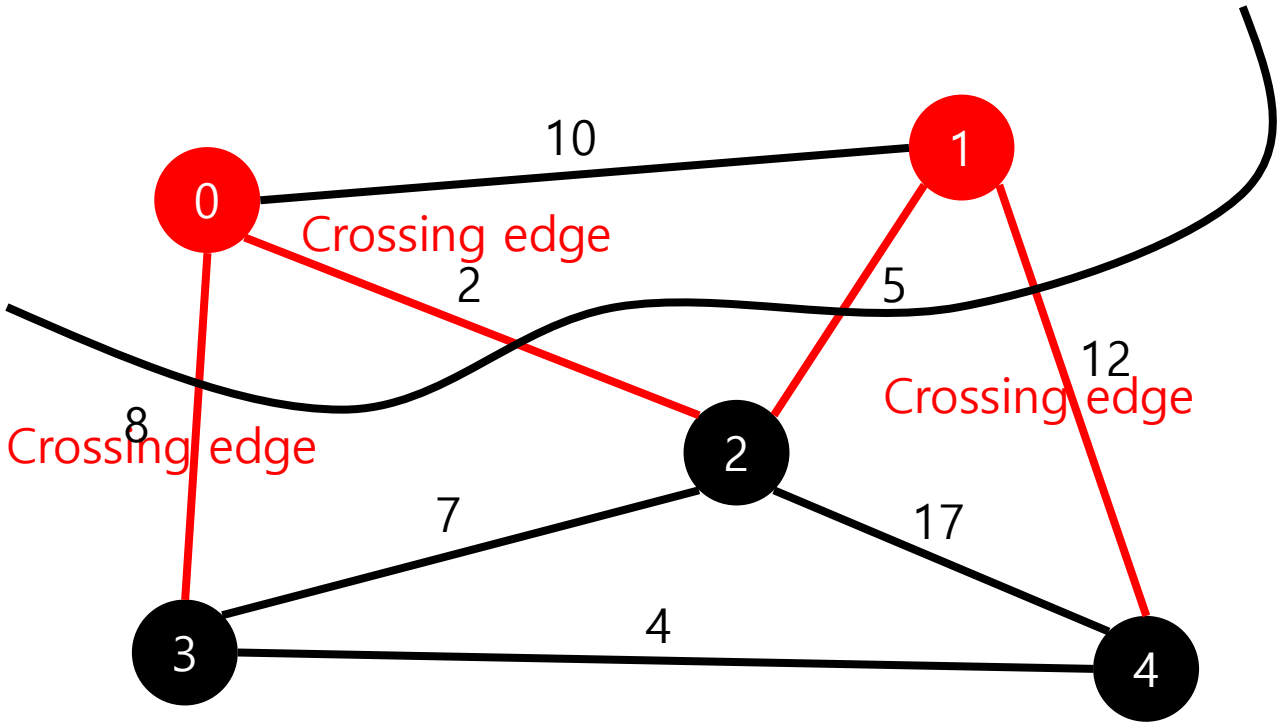
컷(Cut)

V(G) 집합을
공집합이 아닌
두 집합으로 나눈 것



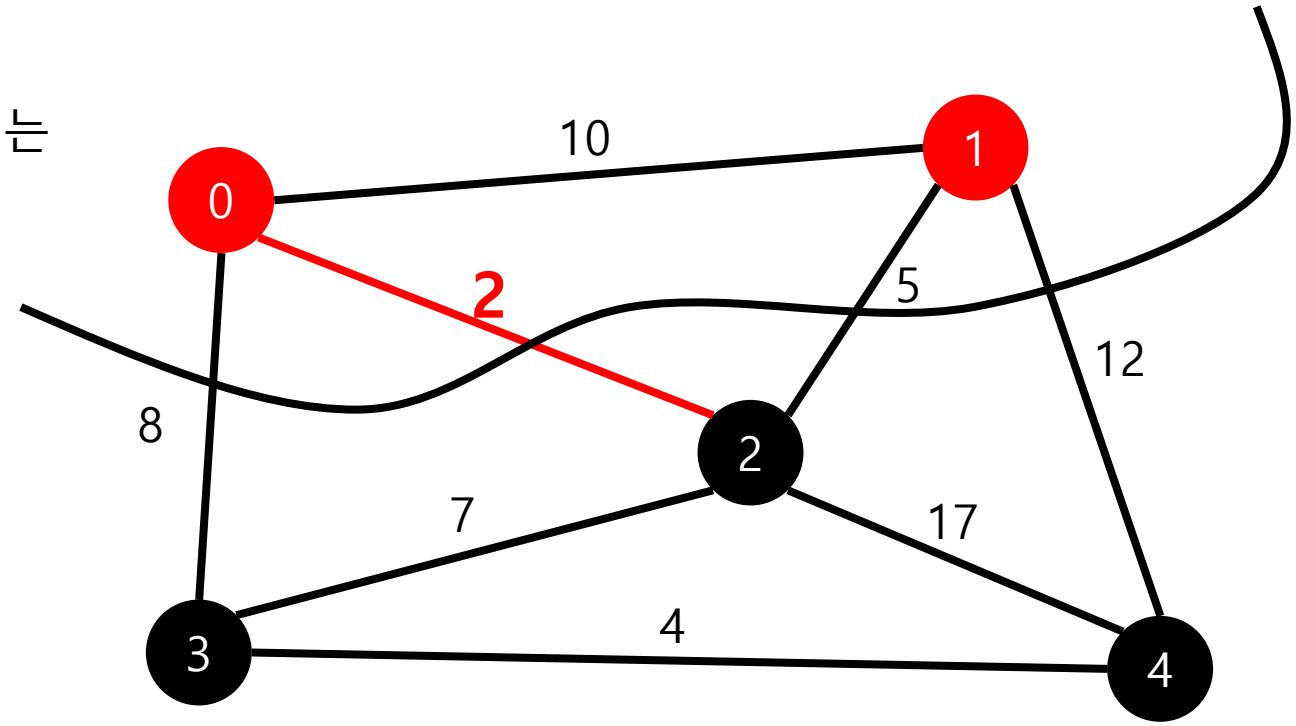
컷(Cut)

Crossing edge는 컷으로 나뉜 한 집합의 정점과 다른 집합의 정점을 잇는다.



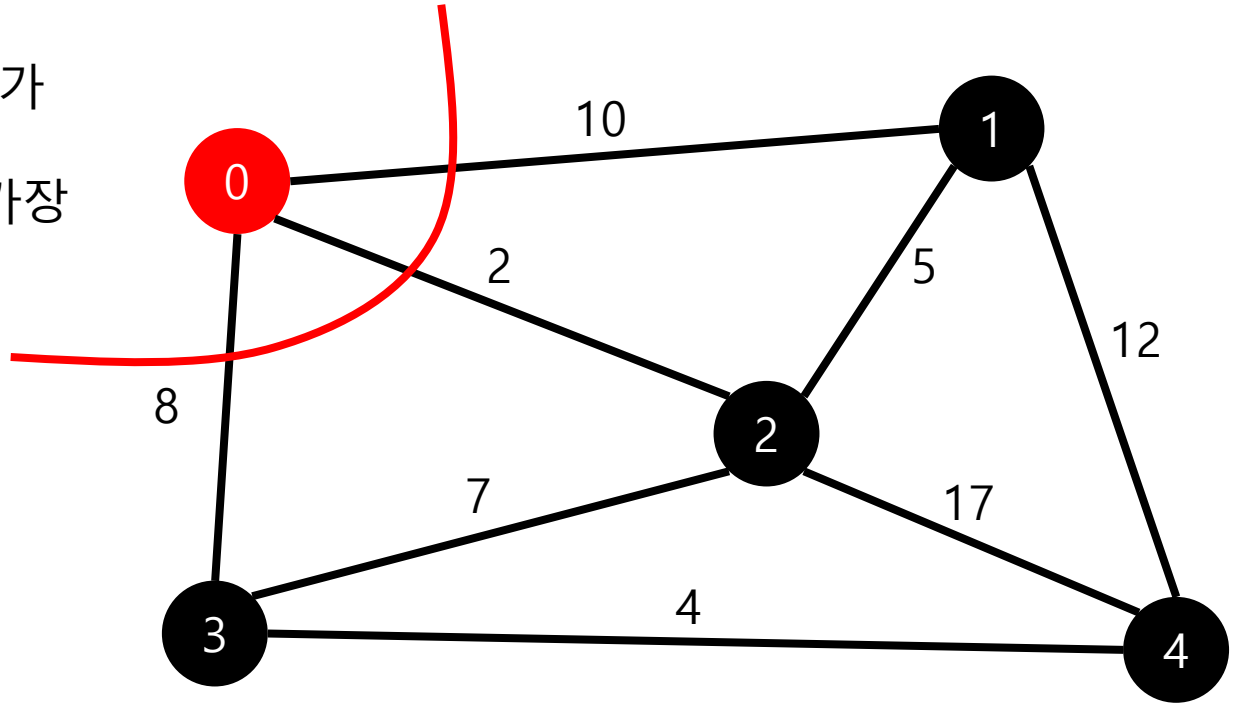
컷 프로퍼티(Cut property)

Crossing edge 중 가장 작은 에지는 MST의 한 에지이다.



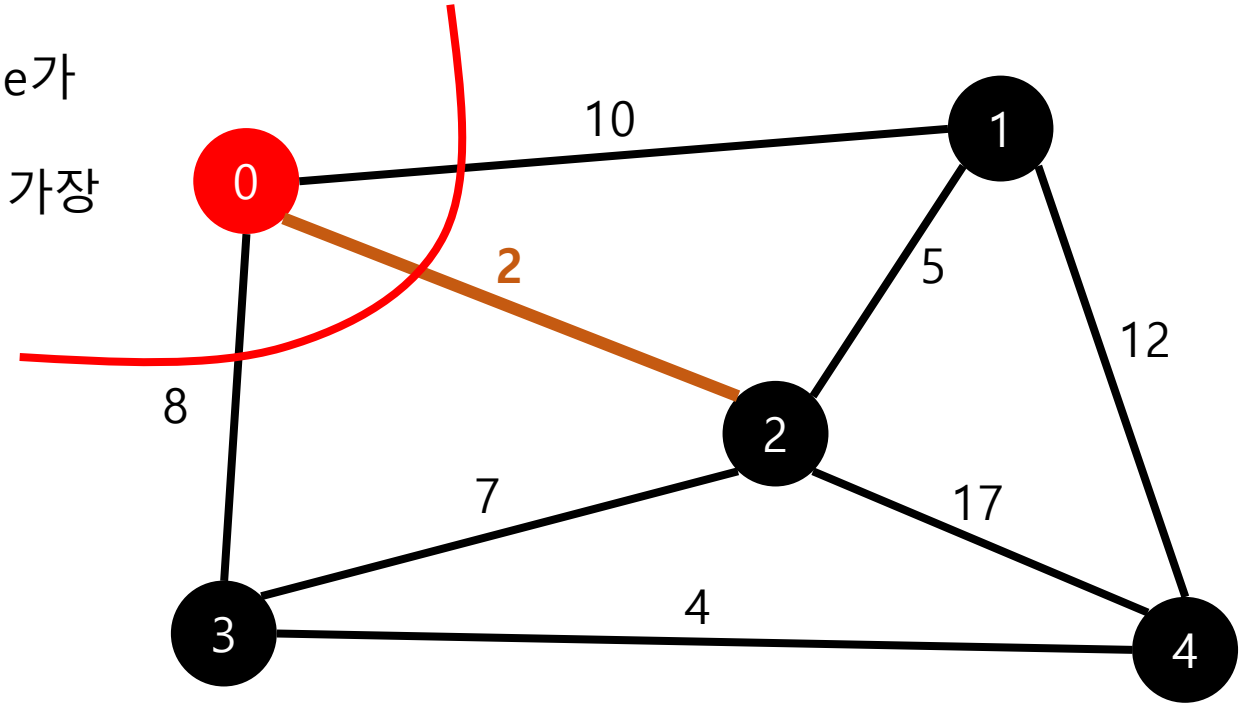
Greedy MST algorithm

1. 선택된 에지가 crossing edge가 되지 않는 컷을 찾은 후
2. Crossing edge 중 가중치가 가장 작은 에지를 찾아 선택한다.



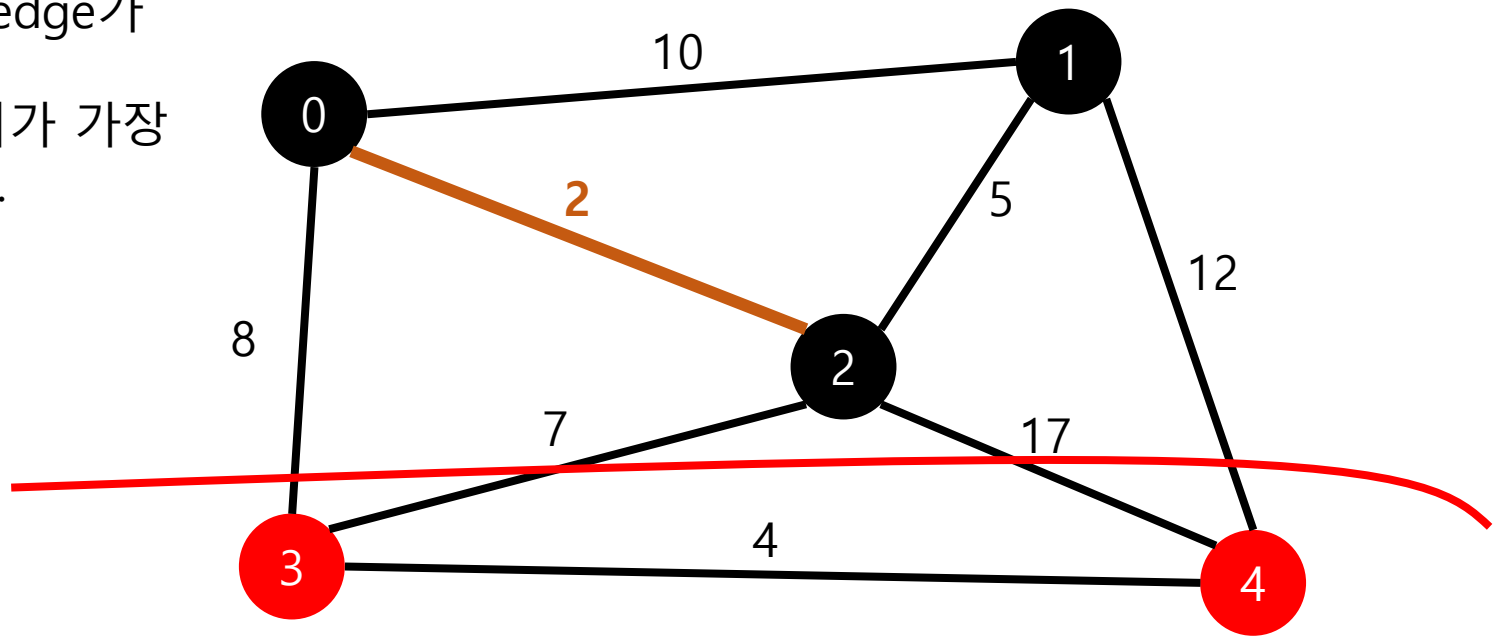
Greedy MST algorithm

1. 선택된 에지가 crossing edge가 되지 않는 컷을 찾은 후
2. Crossing edge 중 가중치가 가장 작은 에지를 찾아 선택한다.



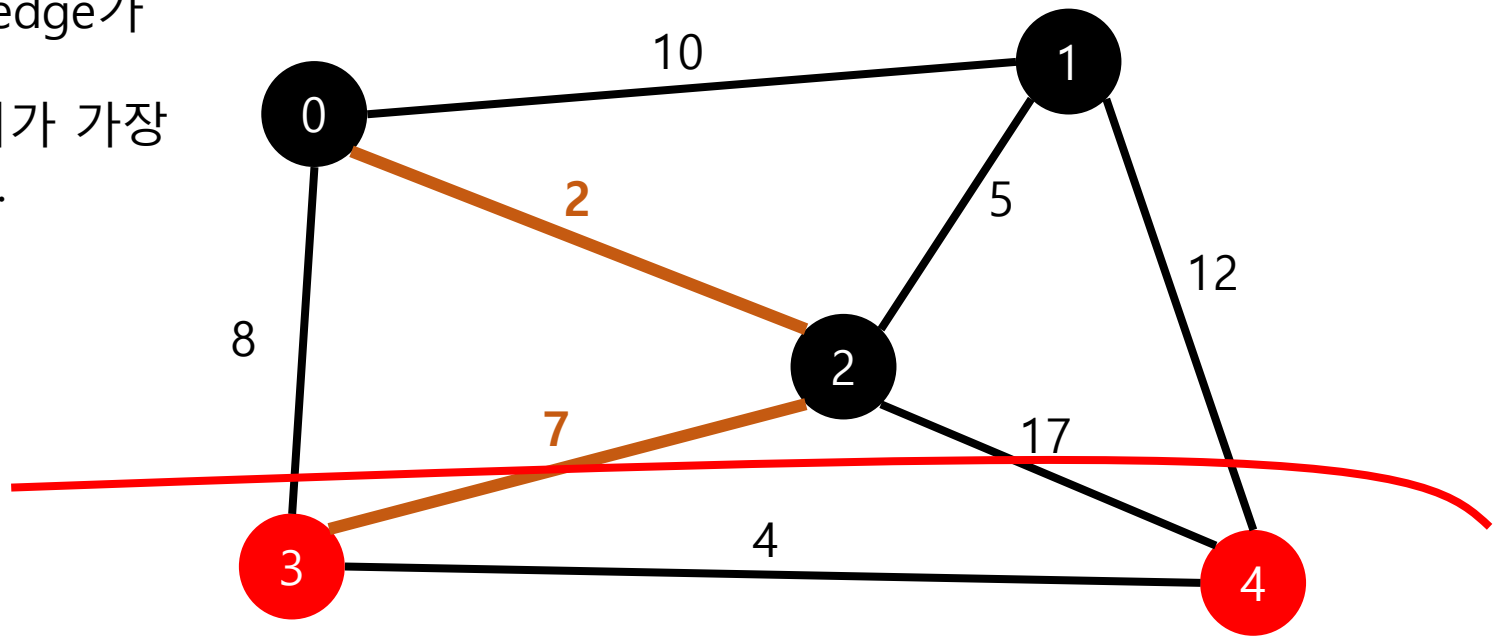
Greedy MST algorithm

1. 선택된 에지가 crossing edge가 되지 않는 컷을 찾은 후
2. Crossing edge 중 가중치가 가장 작은 에지를 찾아 선택한다.



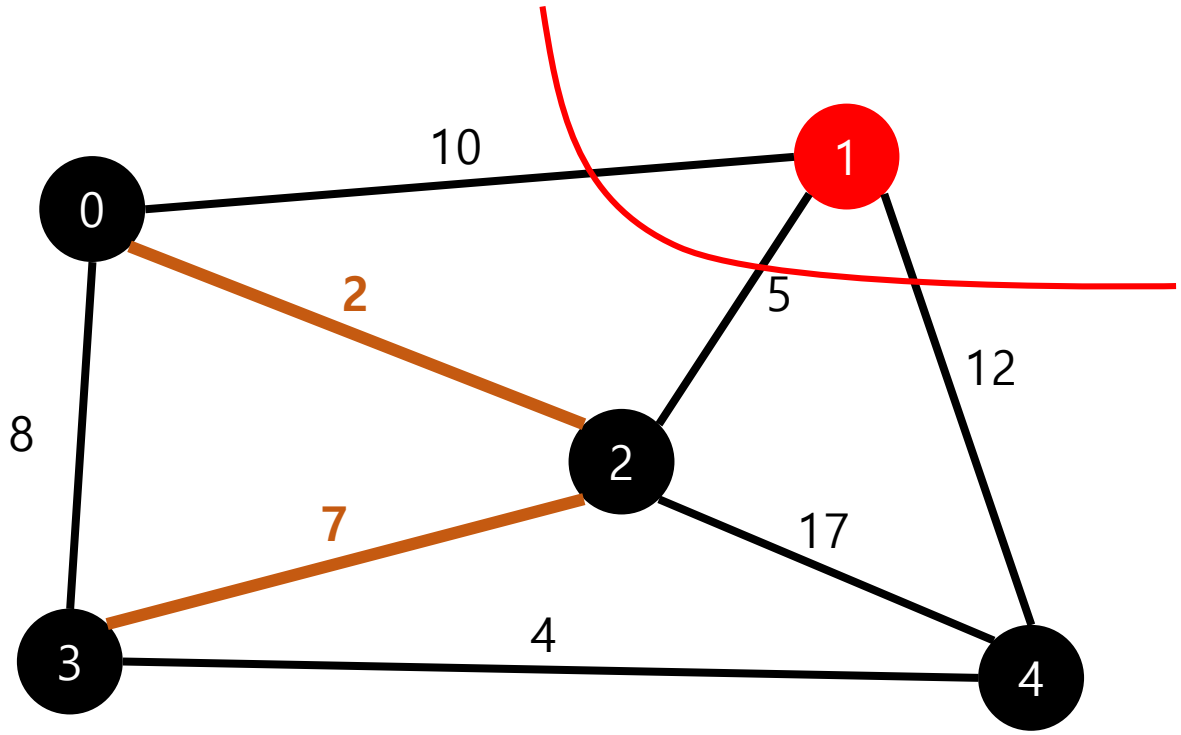
Greedy MST algorithm

1. 선택된 에지가 crossing edge가 되지 않는 컷을 찾은 후
2. Crossing edge 중 가중치가 가장 작은 에지를 찾아 선택한다.



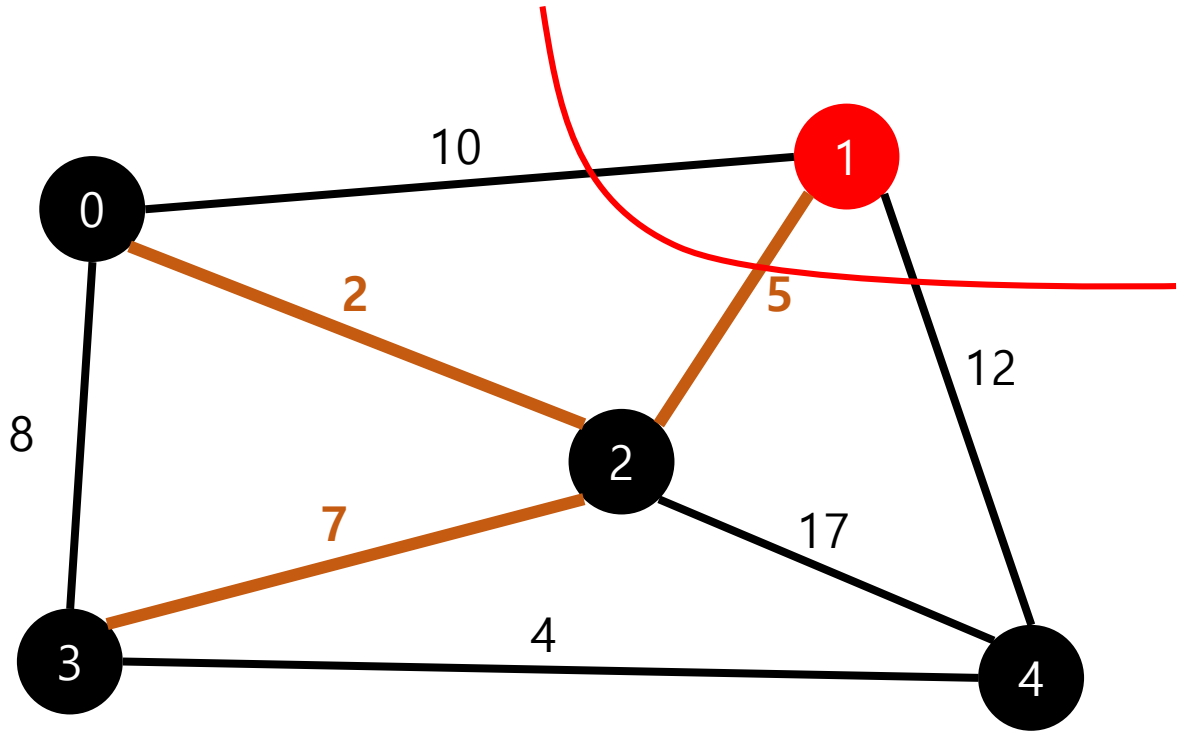
Greedy MST algorithm

1. 선택된 에지가 crossing edge가 되지 않는 컷을 찾은 후
2. Crossing edge 중 가중치가 가장 작은 에지를 찾아 선택한다.



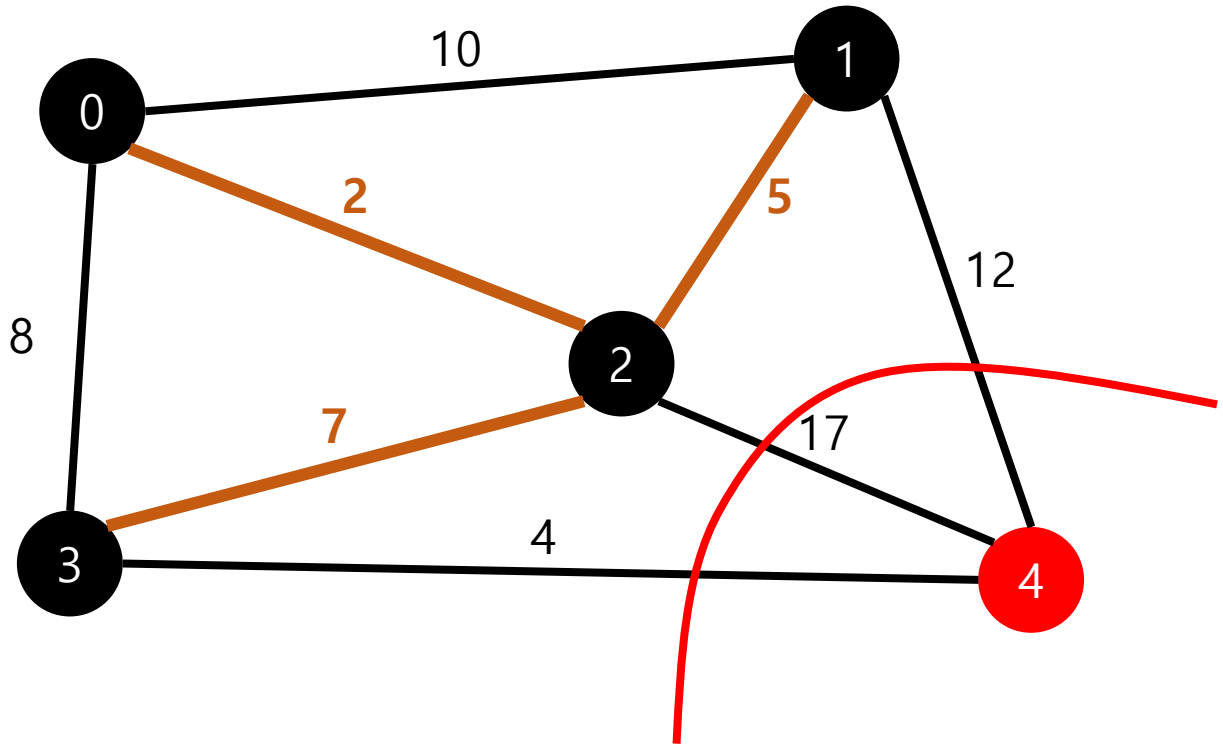
Greedy MST algorithm

1. 선택된 에지가 crossing edge가 되지 않는 컷을 찾은 후
2. Crossing edge 중 가중치가 가장 작은 에지를 찾아 선택한다.



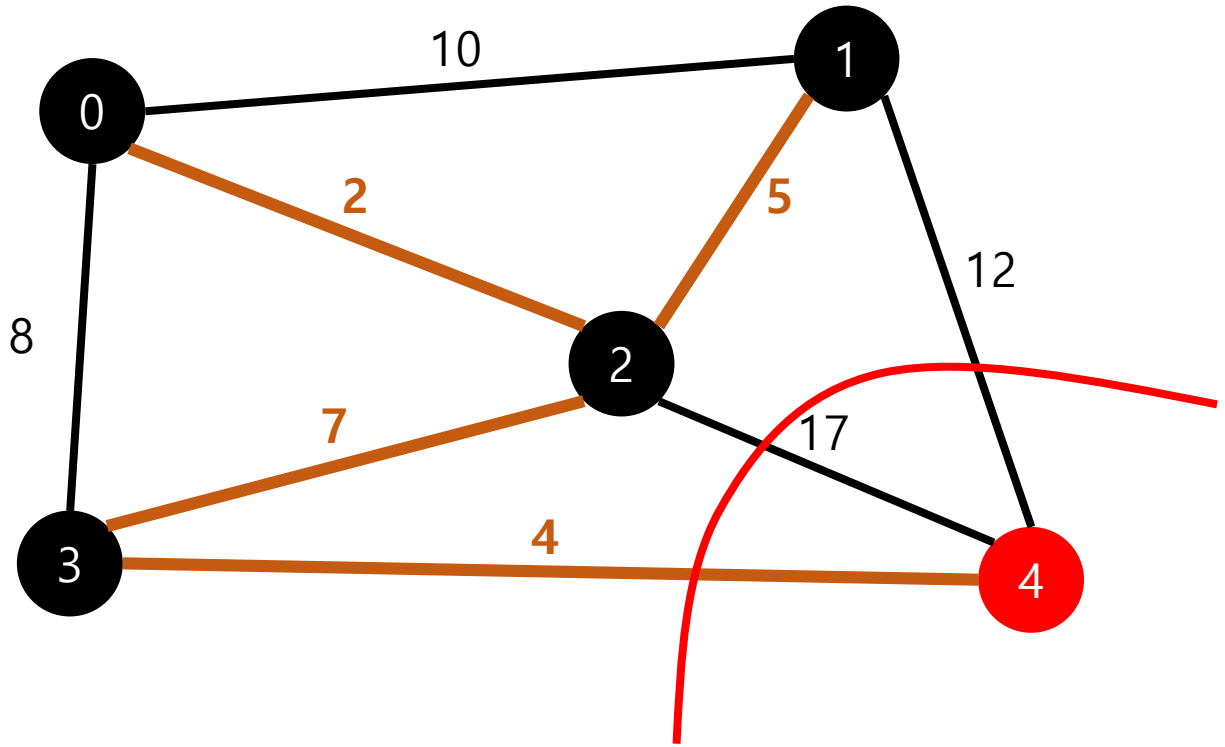
Greedy MST algorithm

1. 선택된 에지가 crossing edge가 되지 않는 컷을 찾은 후
2. Crossing edge 중 가중치가 가장 작은 에지를 찾아 선택한다.

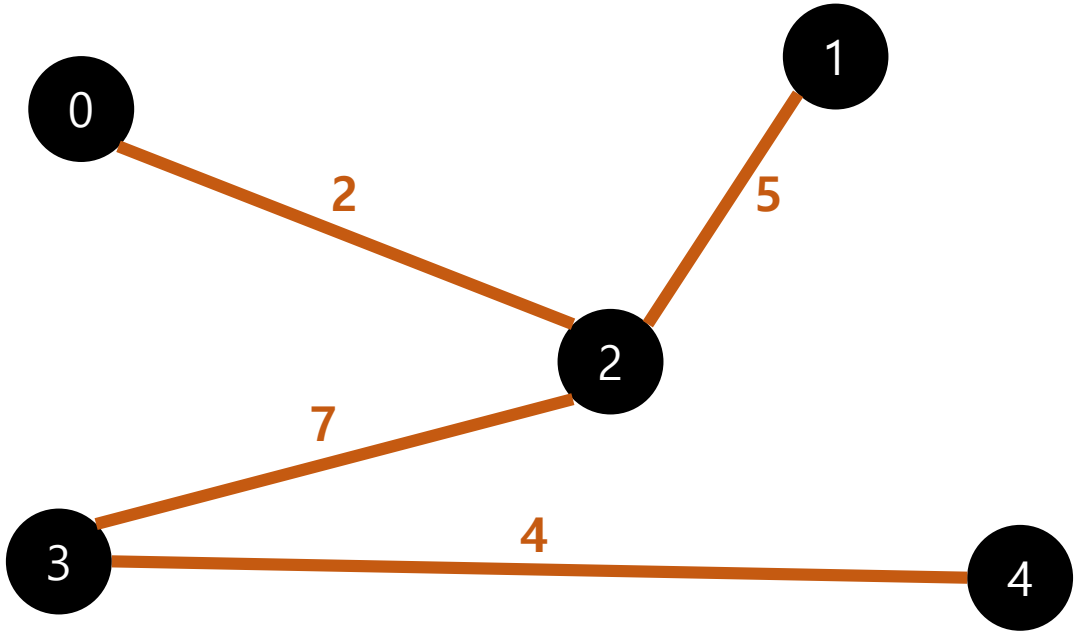


Greedy MST algorithm

1. 선택된 에지가 crossing edge가 되지 않는 컷을 찾은 후
2. Crossing edge 중 가중치가 가장 작은 에지를 찾아 선택한다.

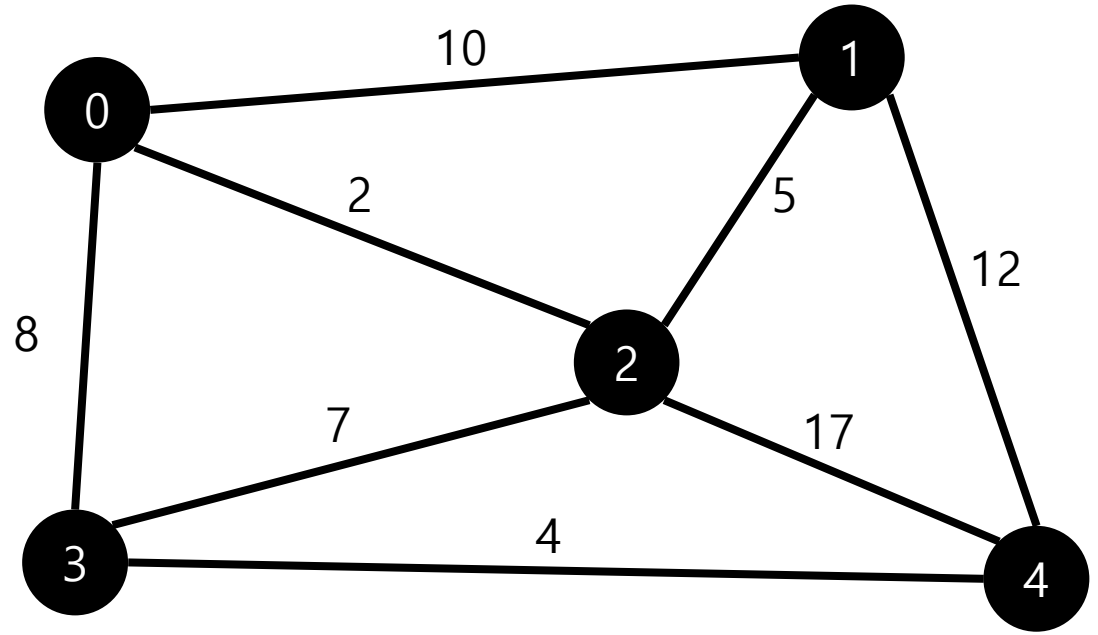


Greedy MST algorithm



Greedy MST algorithm

1. 어떻게 **컷**을 선택할 것인가?
2. 어떻게 가중치가 **가장 작은 에지**를 찾을 것인가?



Kruskal algorithm

Kruskal algorithm

1. 에지를 가중치가 작은 것에서 큰 것 순으로 정렬
2. 트리에 에지를 하나씩 추가
3. 사이클이 생기면 추가하지 않는다.
4. 최소 비용 신장 트리가 완성되면 |E|=|V|-1

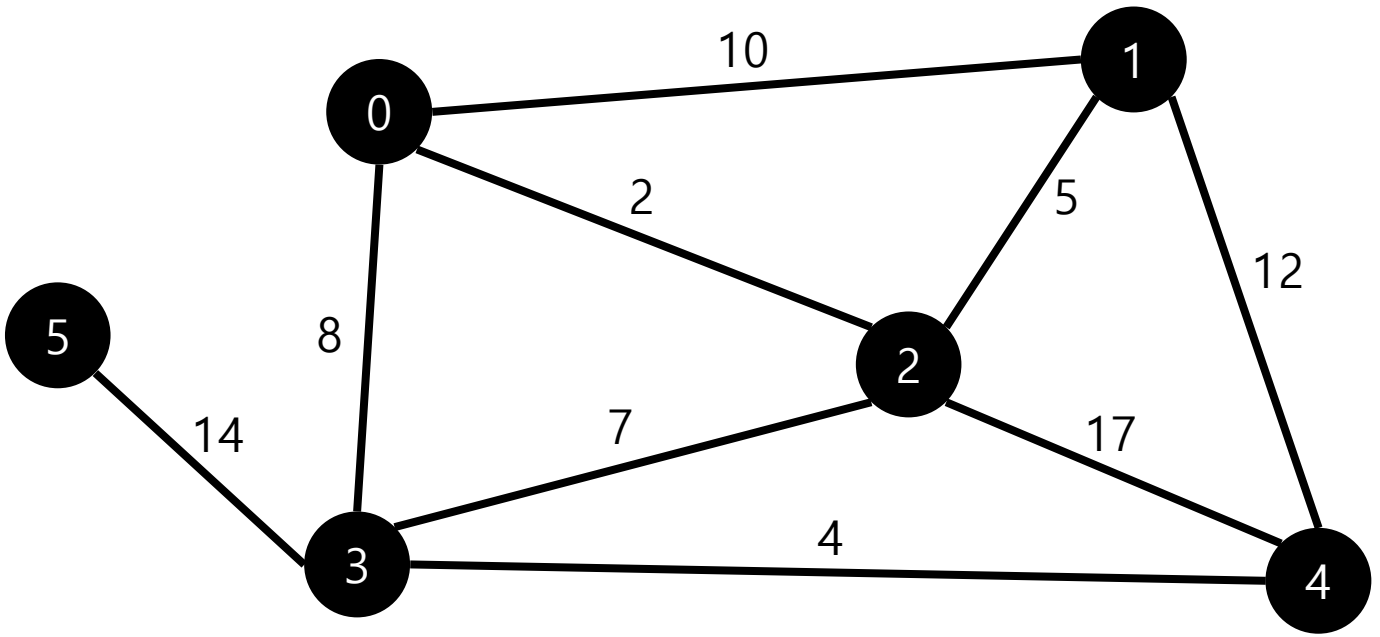
Kruskal algorithm

Sort edges by weight

(0, 2) : 2
(3, 4) : 4
(1, 2) : 5
(3, 2) : 7
(0, 3) : 8
(0, 1) : 10
(1, 4) : 12
(3, 5) : 14
(2, 4) : 17

사이클 형성을 검사하기 위한 집합들

{0} {1} {2} {3} {4} {5}



Kruskal algorithm

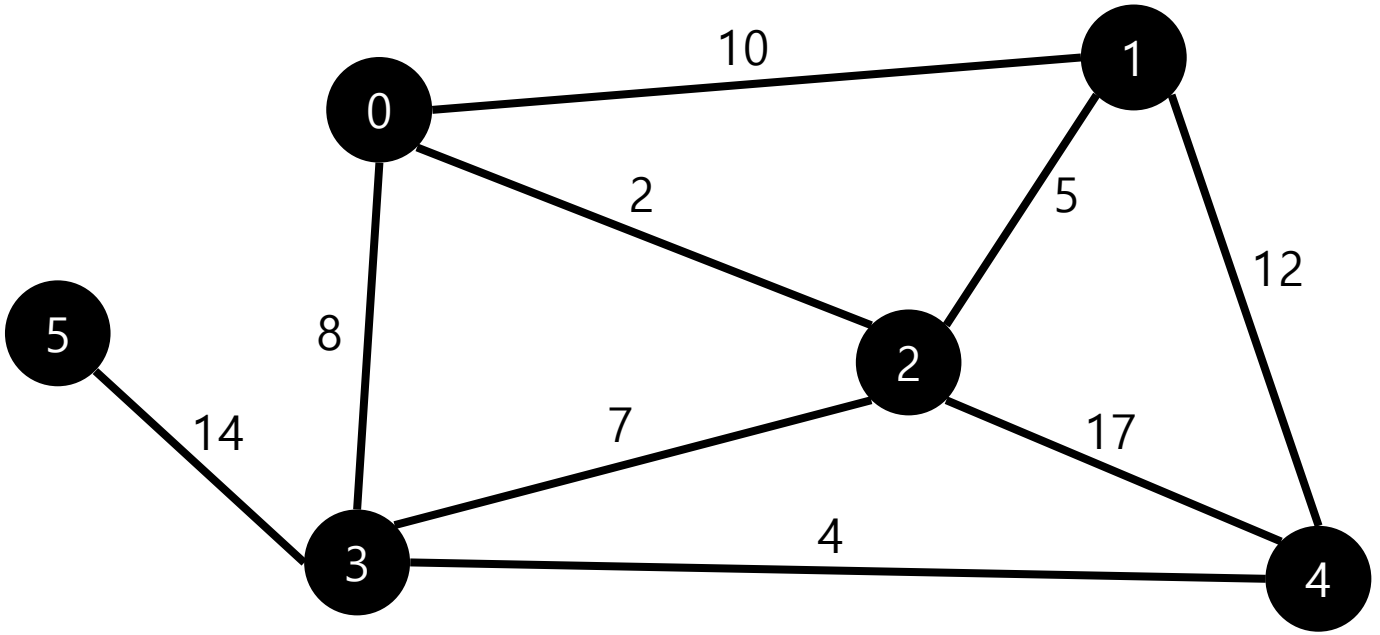
Sort edges by weight

(0, 2) : 2
(3, 4) : 4
(1, 2) : 5
(3, 2) : 7
(0, 3) : 8
(0, 1) : 10
(1, 4) : 12
(3, 5) : 14
(2, 4) : 17

사이클 형성을 검사하기 위한 집합들

{0} {1} {2} {3} {4} {5}

같은 집합 원소가 아니므로 사이클을 형성하지 않는다!!



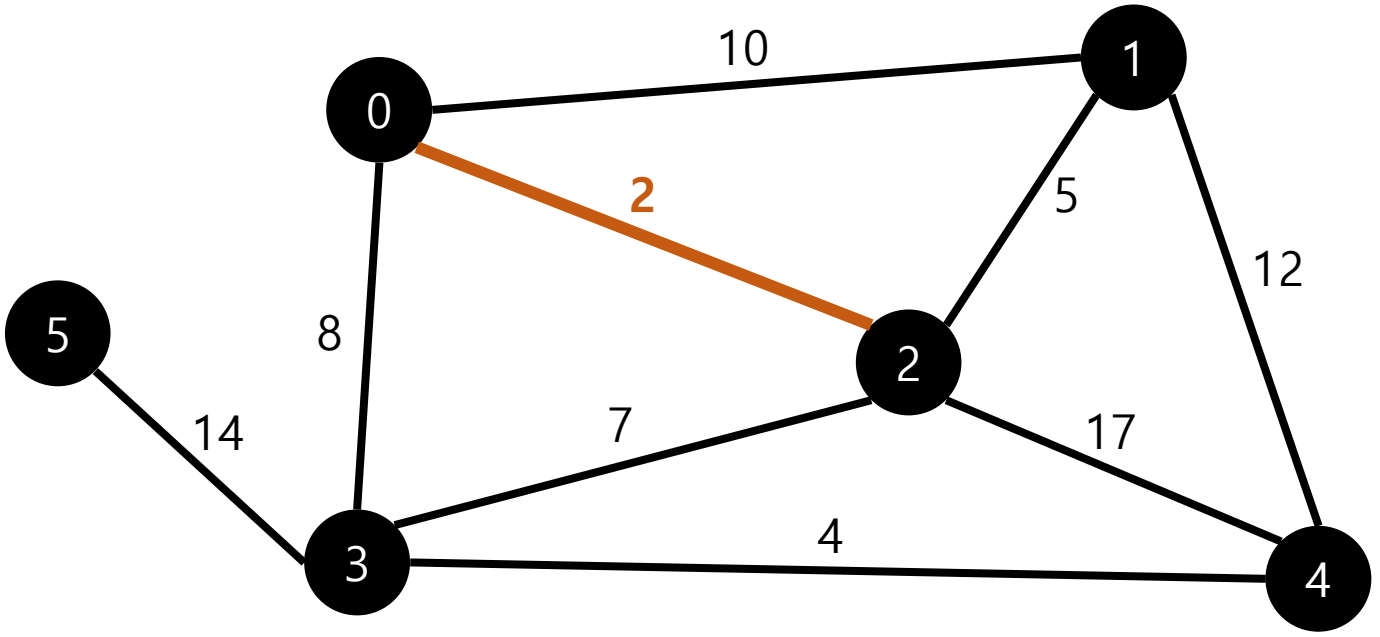
Kruskal algorithm

Sort edges by weight

(0, 2) : 2
(3, 4) : 4
(1, 2) : 5
(3, 2) : 7
(0, 3) : 8
(0, 1) : 10
(1, 4) : 12
(3, 5) : 14
(2, 4) : 17

사이클 형성을 검사하기 위한 집합들

{0, 2} {1} {3} {4} {5}



Kruskal algorithm

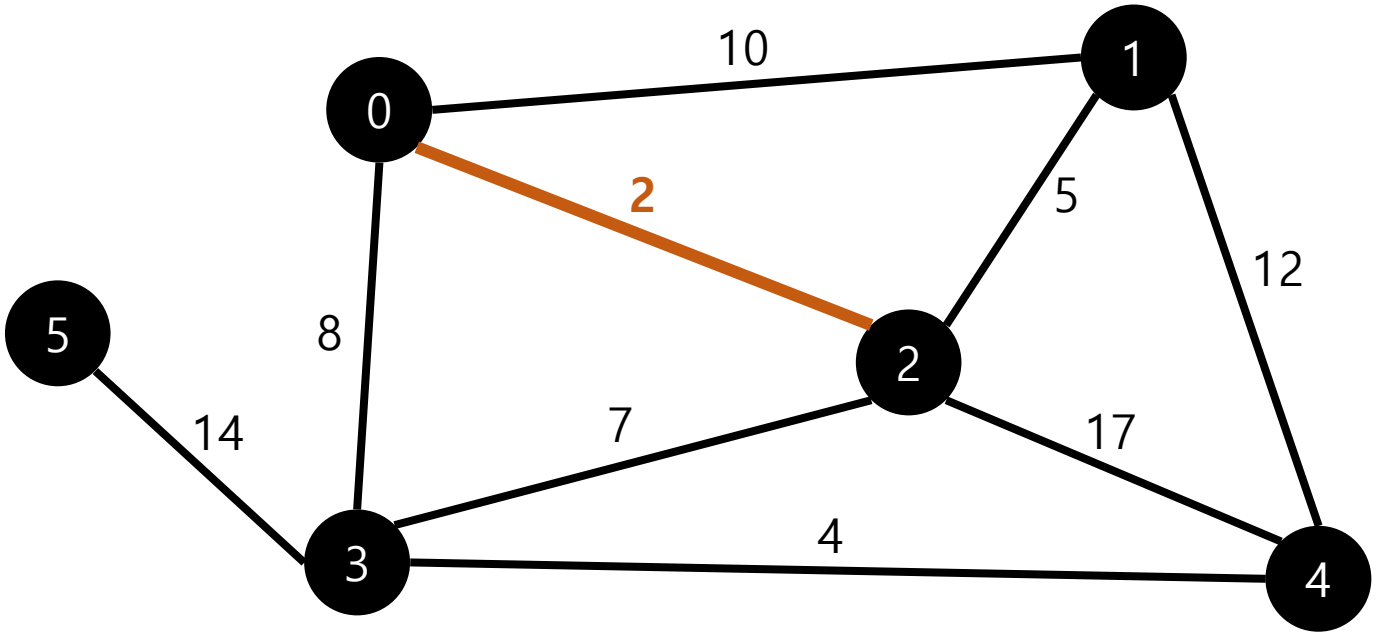
Sort edges by weight

(3, 4) : 4
(1, 2) : 5
(3, 2) : 7
(0, 3) : 8
(0, 1) : 10
(1, 4) : 12
(3, 5) : 14
(2, 4) : 17

사이클 형성을 검사하기 위한 집합들

{0, 2} {1} {3} {4} {5}

같은 집합 원소가 아니므로 사이클을 형성하지 않는다!!



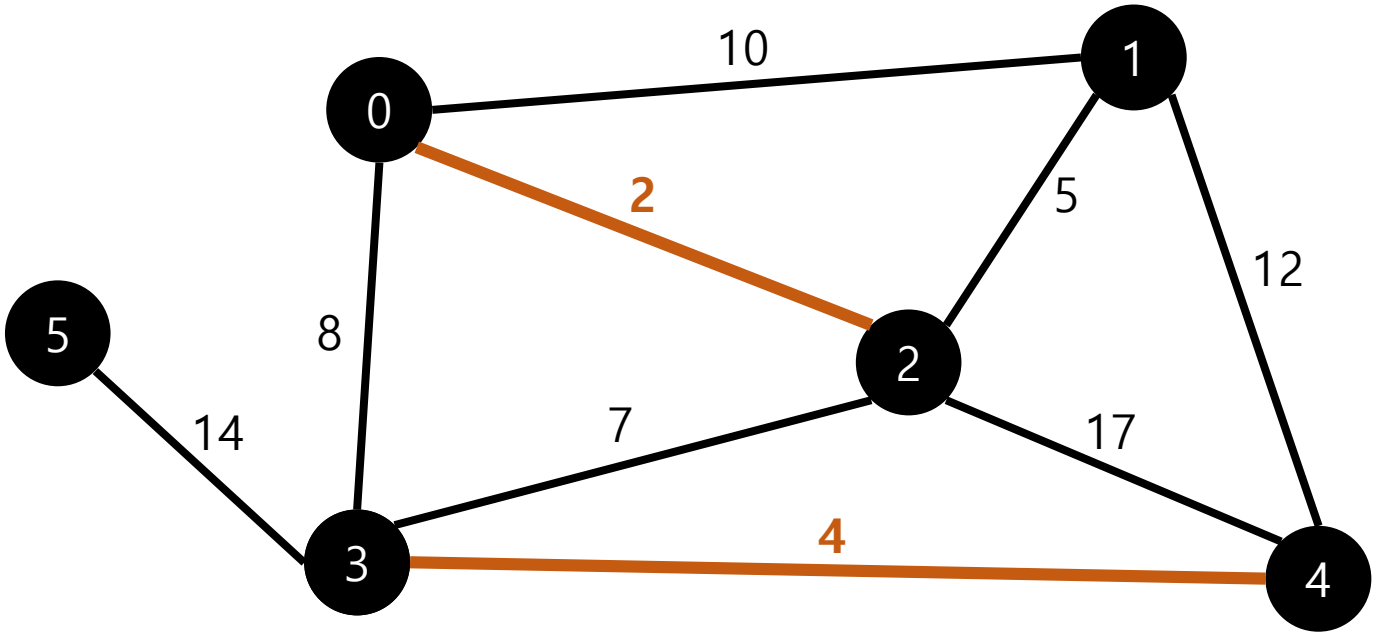
Kruskal algorithm

Sort edges by weight

(3, 4) : 4
(1, 2) : 5
(3, 2) : 7
(0, 3) : 8
(0, 1) : 10
(1, 4) : 12
(3, 5) : 14
(2, 4) : 17

사이클 형성을 검사하기 위한 집합들

{0, 2} {1} {3, 4} {5}



Kruskal algorithm

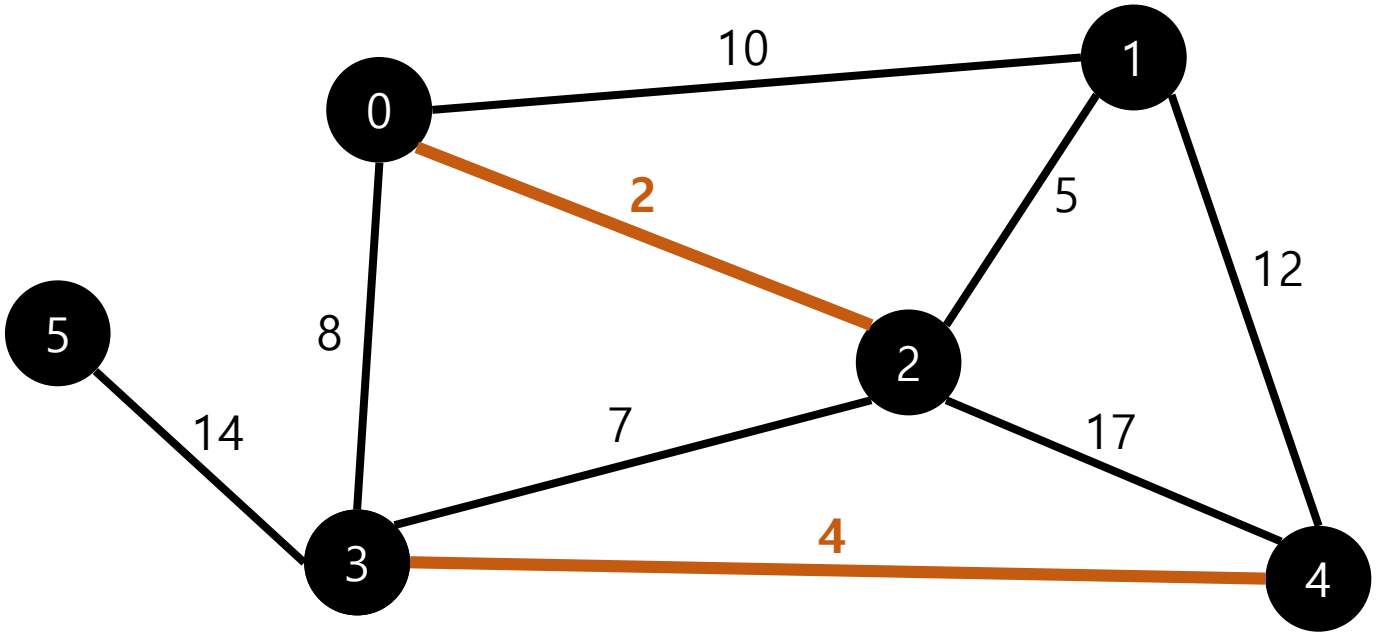
Sort edges by weight

(1, 2) : 5
(3, 2) : 7
(0, 3) : 8
(0, 1) : 10
(1, 4) : 12
(3, 5) : 14
(2, 4) : 17

사이클 형성을 검사하기 위한 집합들

{0, 2} {1} {3, 4} {5}

같은 집합 원소가 아니므로 사이클을 형성하지 않는다!!



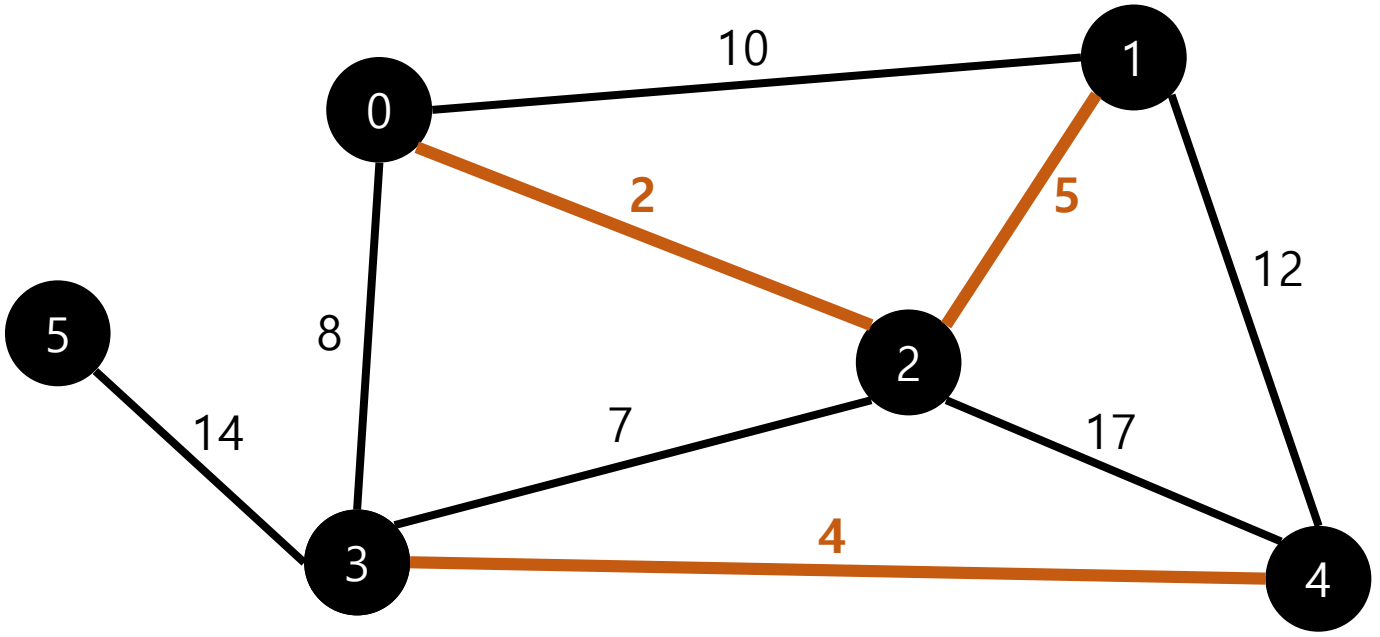
Kruskal algorithm

Sort edges by weight

(1, 2) : 5
(3, 2) : 7
(0, 3) : 8
(0, 1) : 10
(1, 4) : 12
(3, 5) : 14
(2, 4) : 17

사이클 형성을 검사하기 위한 집합들

{0, 1, 2} {3, 4} {5}



Kruskal algorithm

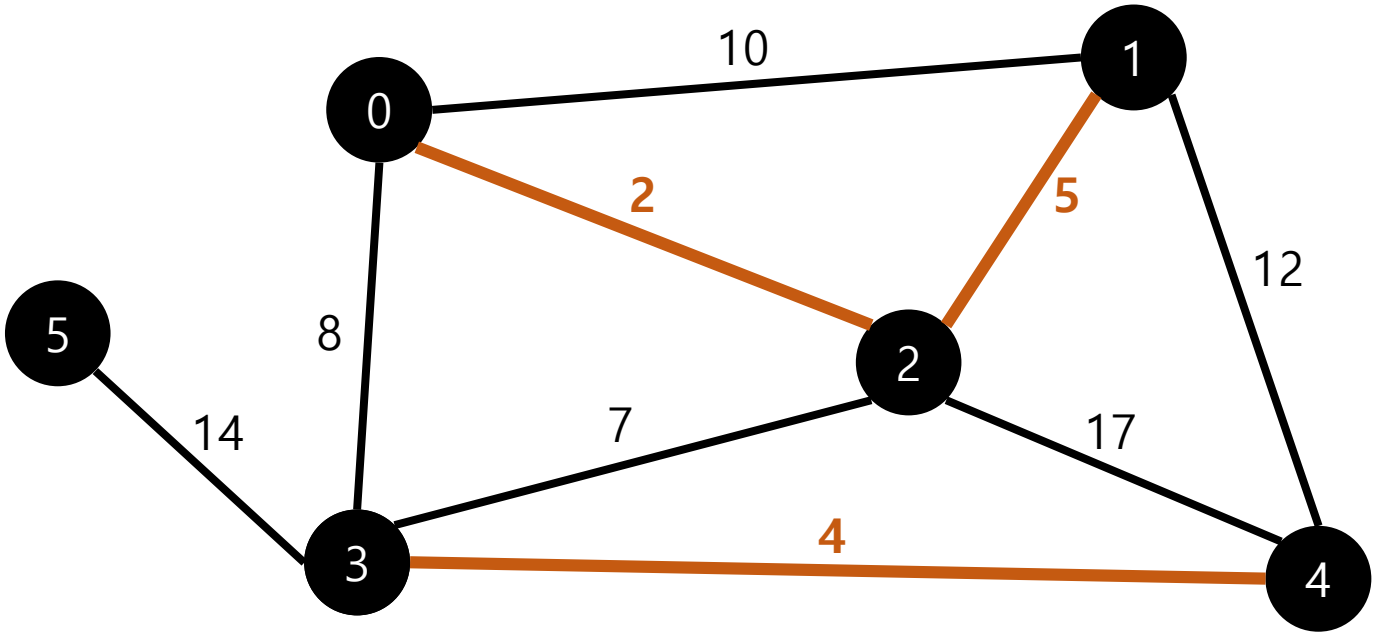
Sort edges by weight

(3, 2) : 7
(0, 3) : 8
(0, 1) : 10
(1, 4) : 12
(3, 5) : 14
(2, 4) : 17

사이클 형성을 검사하기 위한 집합들

{0, 1, 2} {3, 4} {5}

같은 집합 원소가 아니므로 사이클을 형성하지 않는다!!



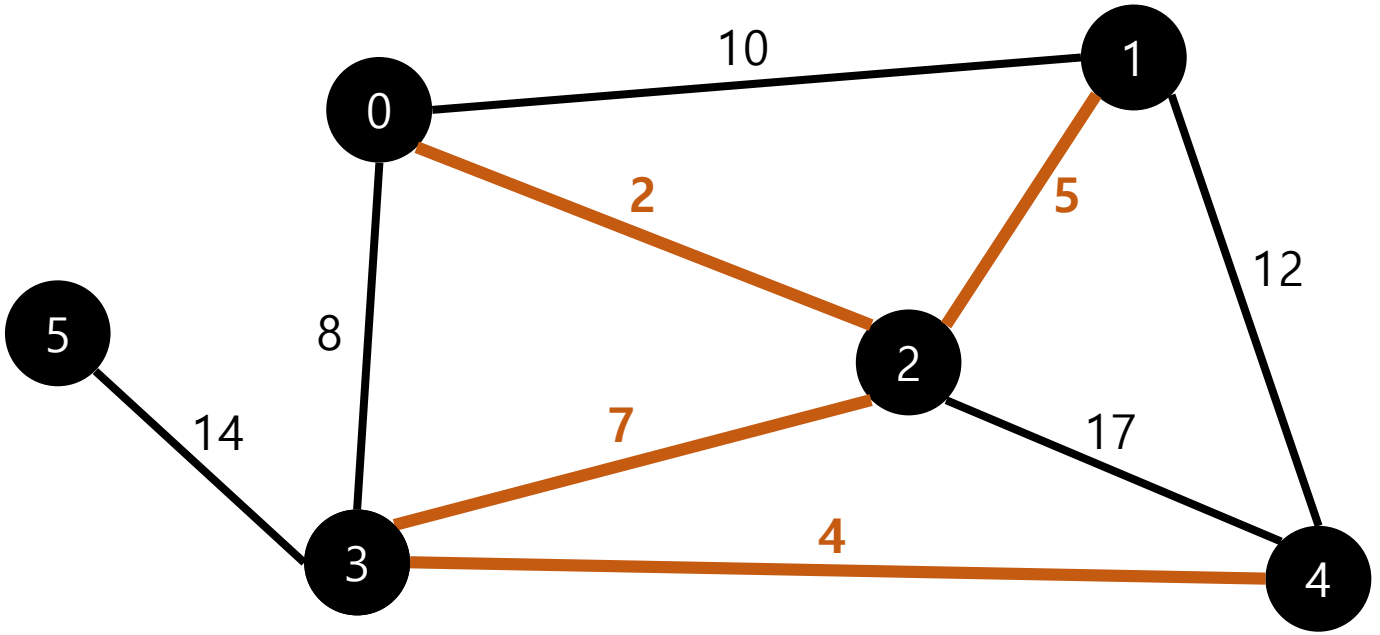
Kruskal algorithm

Sort edges by weight

(3, 2) : 7
(0, 3) : 8
(0, 1) : 10
(1, 4) : 12
(3, 5) : 14
(2, 4) : 17

사이클 형성을 검사하기 위한 집합들

{0, 1, 2, 3, 4} {5}



Kruskal algorithm

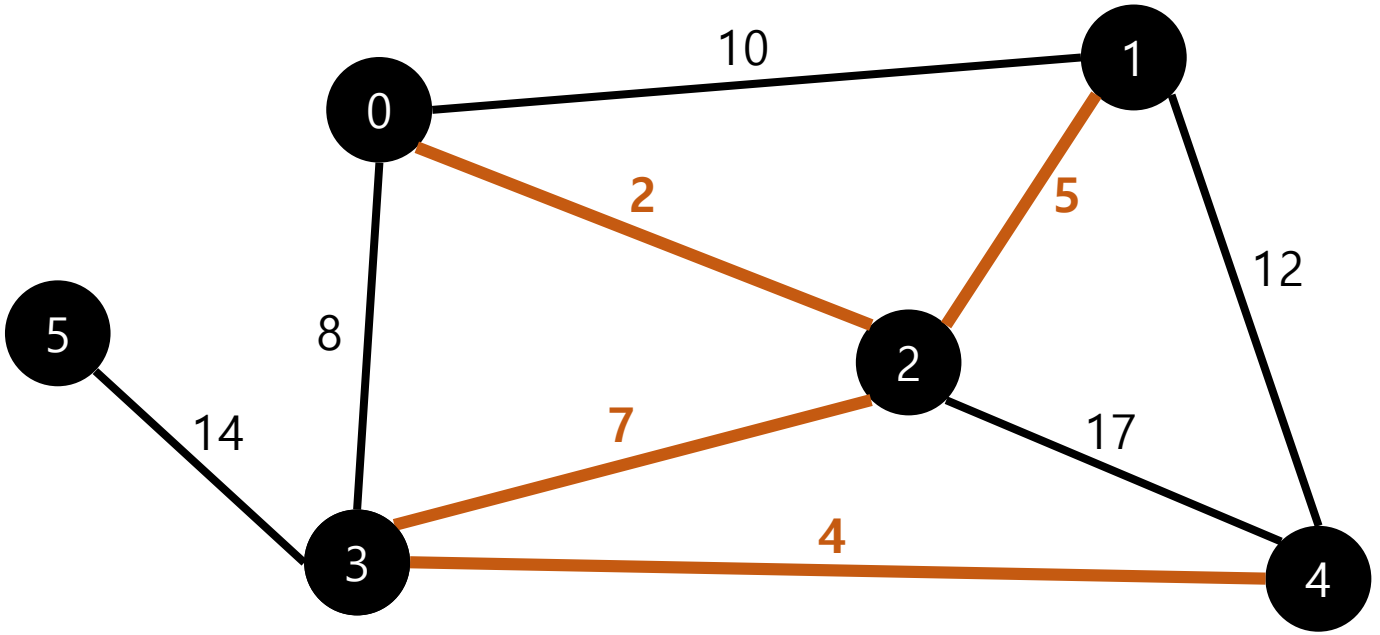
Sort edges by weight

(0, 3) : 8
(0, 1) : 10
(1, 4) : 12
(3, 5) : 14
(2, 4) : 17

사이클 형성을 검사하기 위한 집합들

{0, 1, 2, 3, 4} {5}

사이클을 형성!



Kruskal algorithm

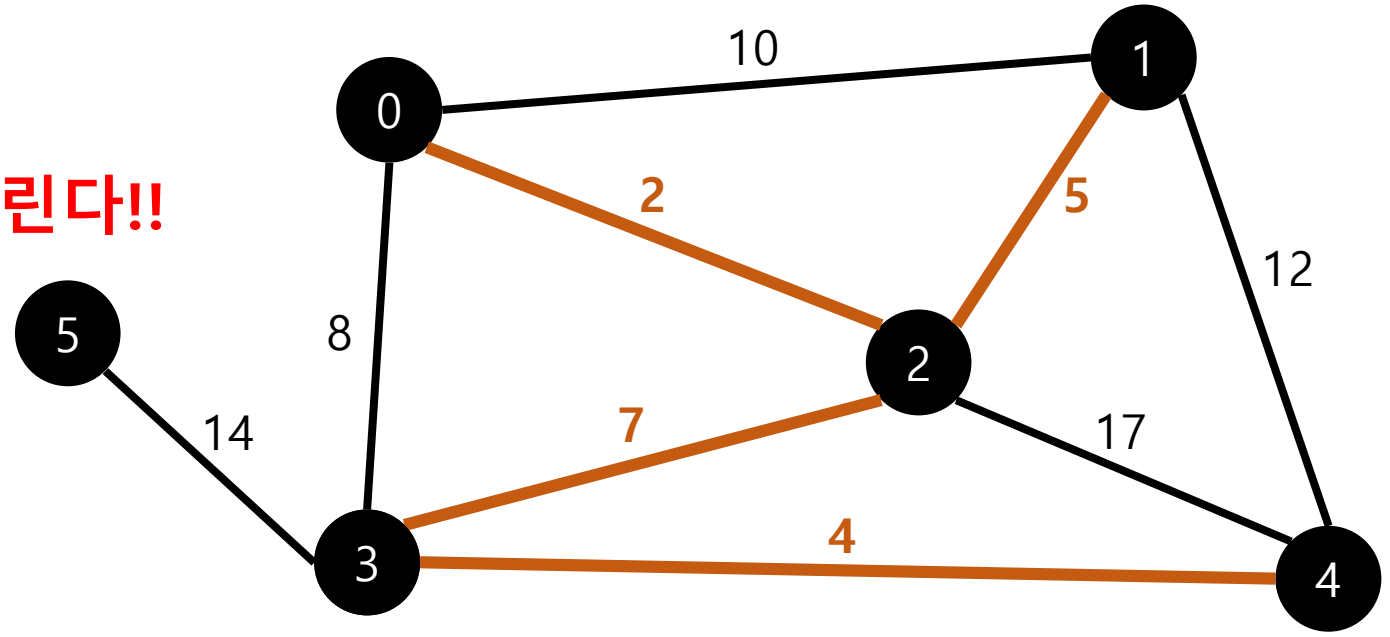
Sort edges by weight

(0, 3) : 8
(0, 1) : 10
(1, 4) : 12
(3, 5) : 14
(2, 4) : 17

선택하지 않고 버린다!!

사이클 형성을 검사하기 위한 집합들

{0, 1, 2, 3, 4} {5}



Kruskal algorithm

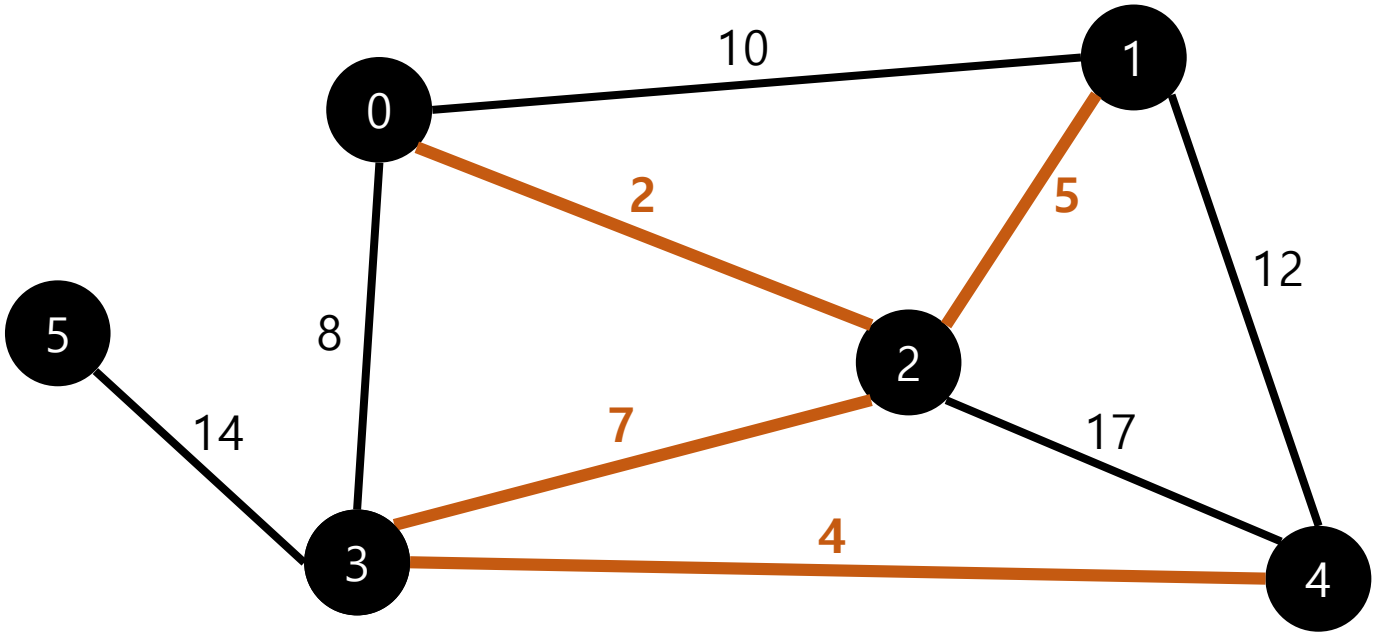
Sort edges by weight

(0, 1) : 10
(1, 4) : 12
(3, 5) : 14
(2, 4) : 17

사이클 형성을 검사하기 위한 집합들

{0, 1, 2, 3, 4} {5}

사이클을 형성!



Kruskal algorithm

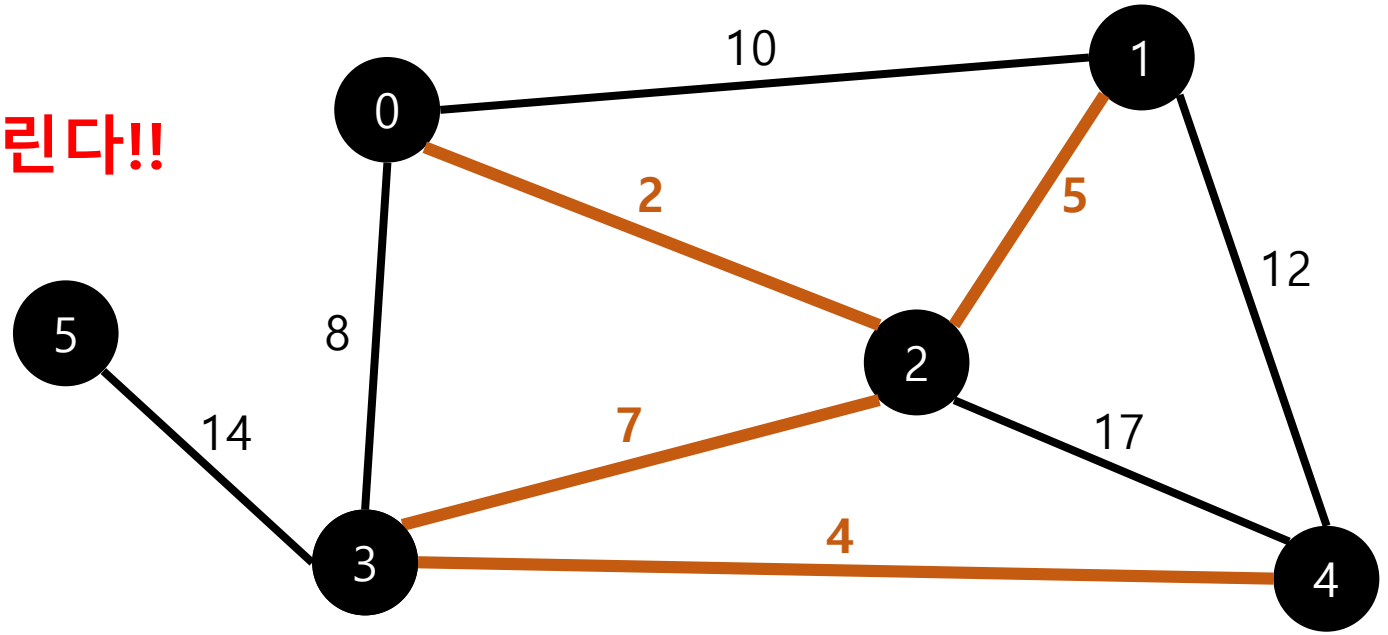
Sort edges by weight

(0, 1) : 10
(1, 4) : 12
(3, 5) : 14
(2, 4) : 17

선택하지 않고 버린다!!

사이클 형성을 검사하기 위한 집합들

{0, 1, 2, 3, 4} {5}



Kruskal algorithm

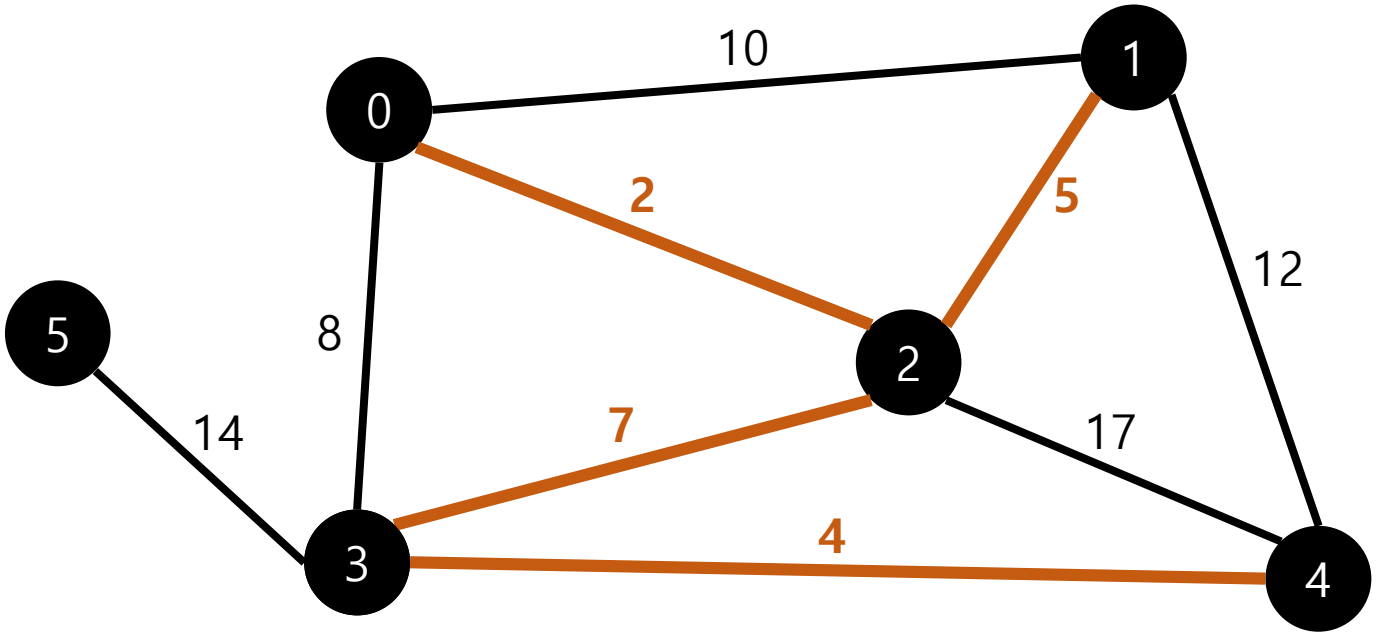
Sort edges by weight

(1, 4) : 12
(3, 5) : 14
(2, 4) : 17

사이클 형성을 검사하기 위한 집합들

{0, 1, 2, 3, 4} {5}

사이클을 형성!



Kruskal algorithm

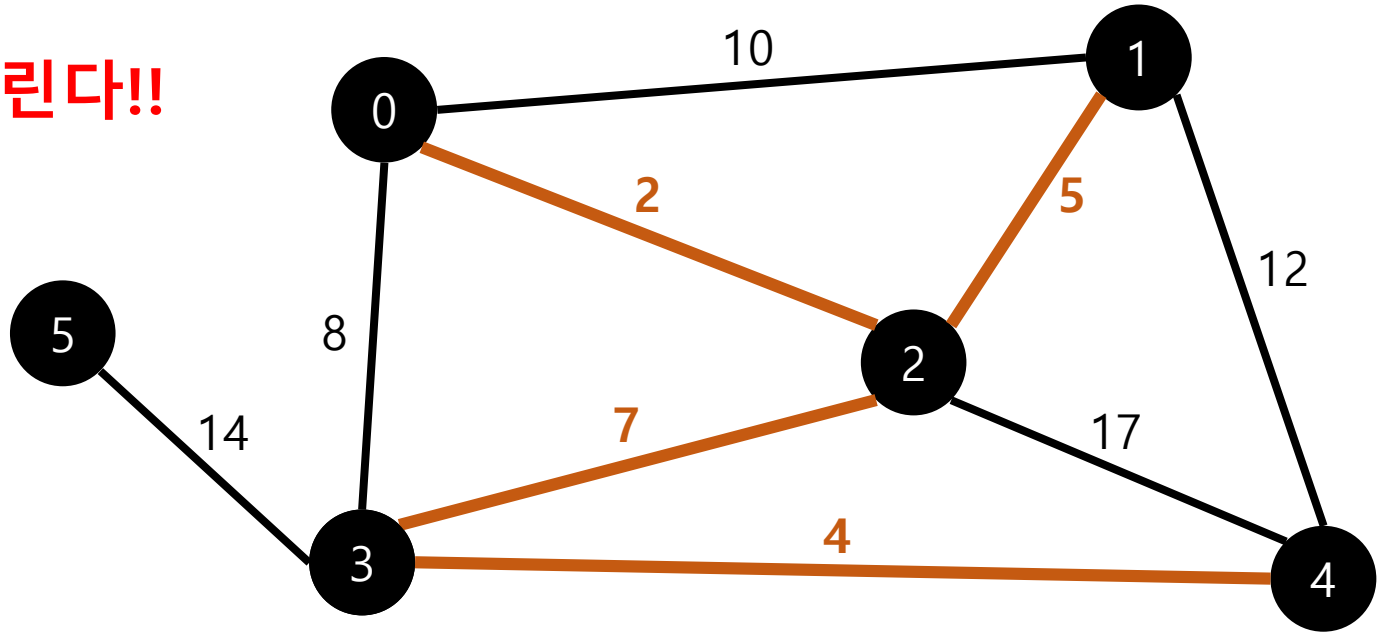
Sort edges by weight

(1, 4) : 12
(3, 5) : 14
(2, 4) : 17

선택하지 않고 버린다!!

사이클 형성을 검사하기 위한 집합들

{0, 1, 2, 3, 4} {5}



Kruskal algorithm

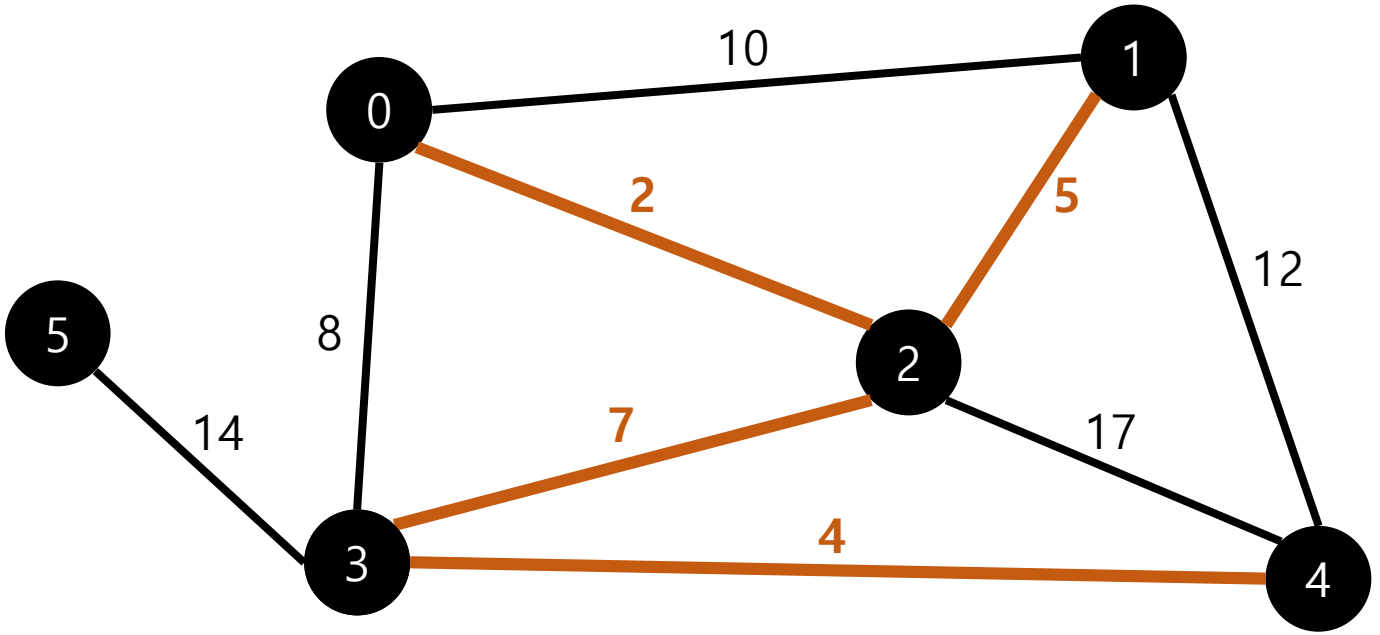
Sort edges by weight

(3, 5) : 14
(2, 4) : 17

사이클 형성을 검사하기 위한 집합들

{0, 1, 2, 3, 4} {5}

같은 집합 원소가 아니므로 사이클을 형성하지 않는다!!



Kruskal algorithm

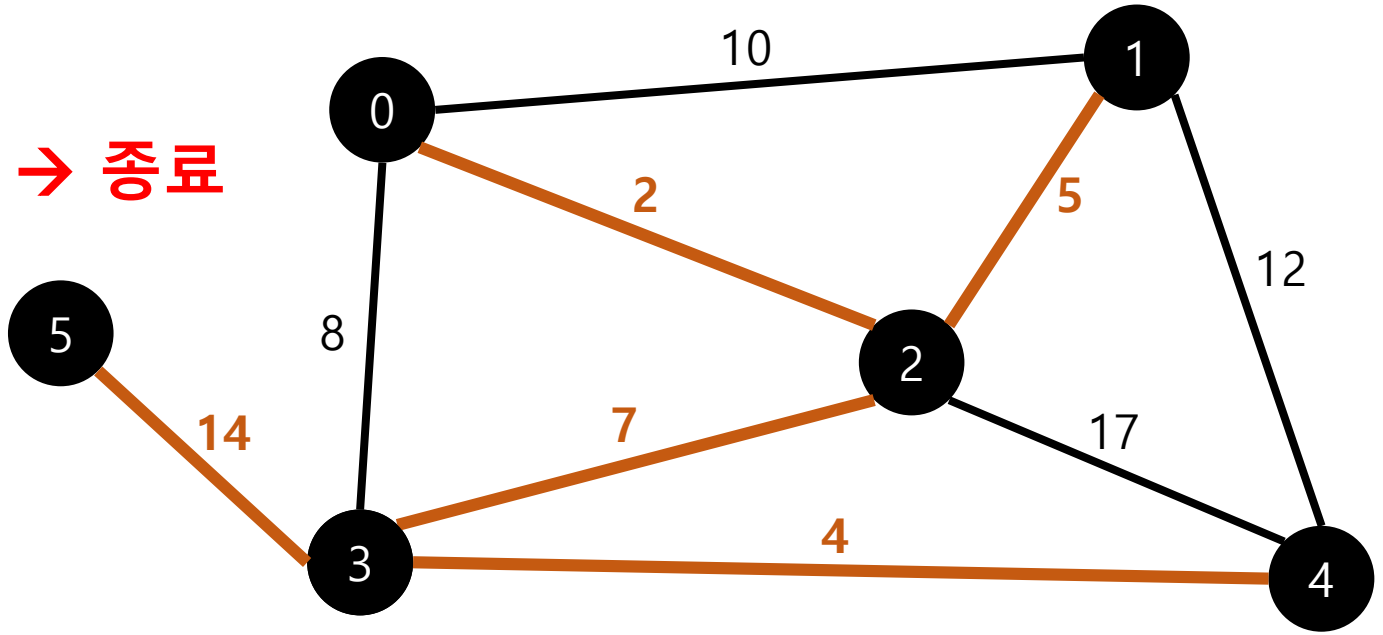
Sort edges by weight

(3, 5) : 14
(2, 4) : 17

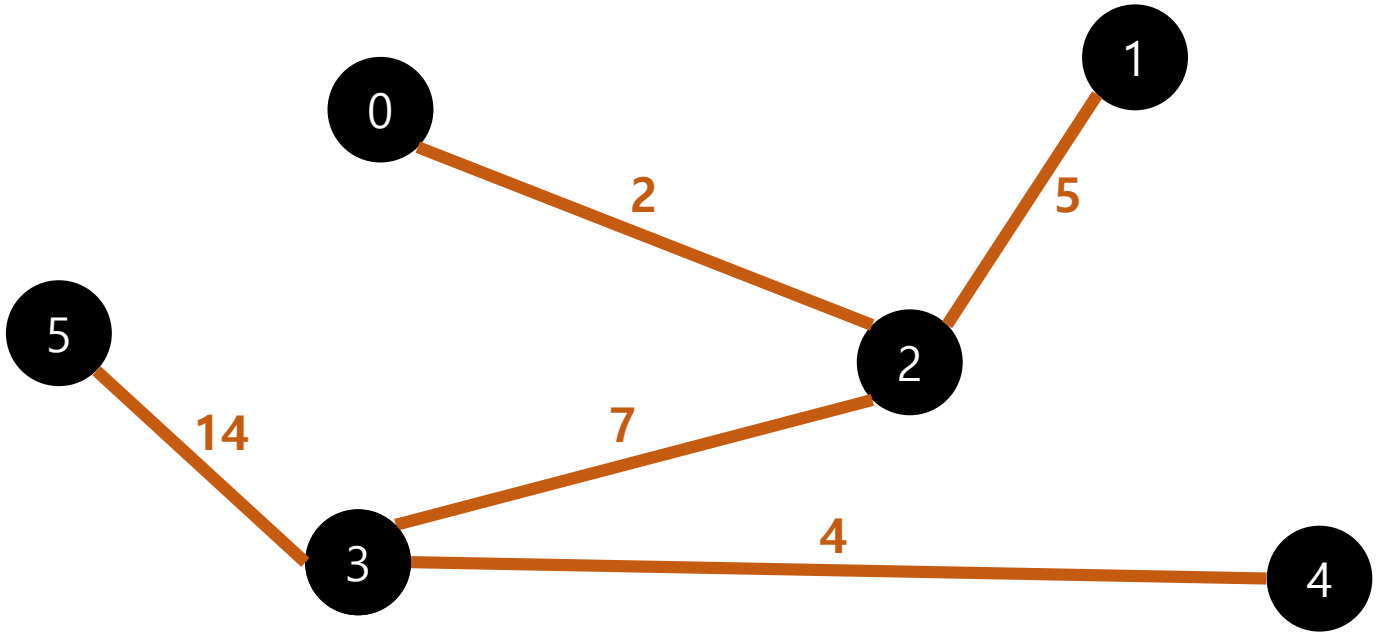
사이클 형성을 검사하기 위한 집합들

{0, 1, 2, 3, 4, 5}

|E|=|V|-1 → 종료



Kruskal algorithm



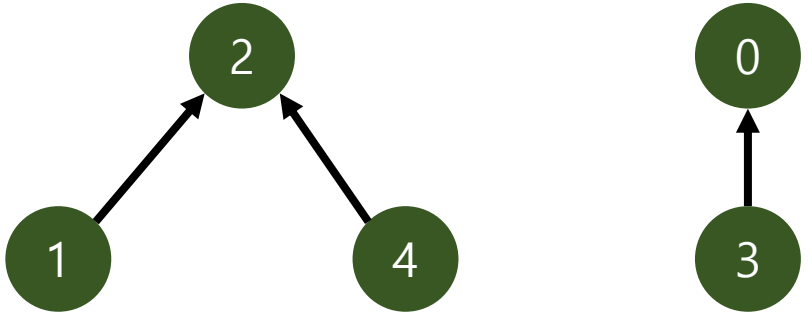
분리 집합(disjoint set)

i	**0**	1	**2**	3	4
parent	-1	2	-1	0	2

{1, 2, 4} {0, 3}

2가지 연산

1. **FIND**
2. **UNION**

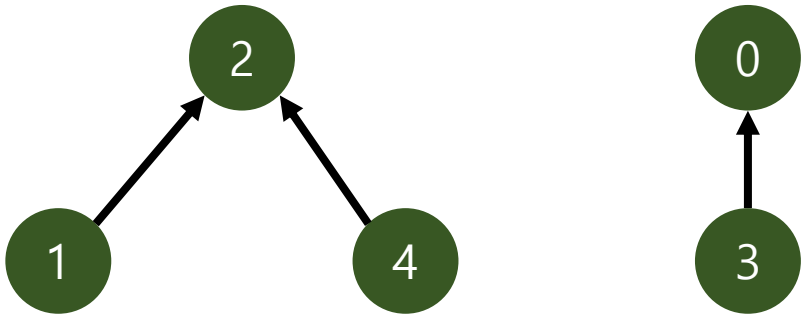


분리 집합 : FIND

i	0	1	2	3	4
parent	-1	2	-1	0	2

FIND(i)
: 정점 i가 포함된 트리의 루트를 찾아 반환

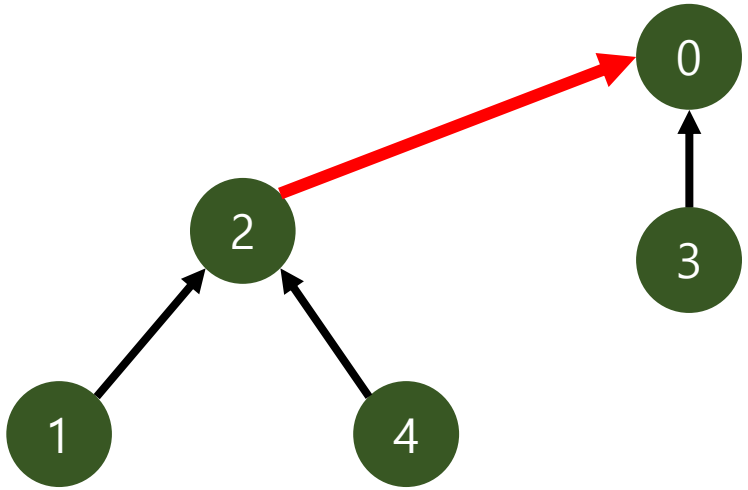
FIND(1) → 2



분리 집합 : UNION

i	0	1	2	3	4
parent	-1	2	**0**	0	2

UNION(i, j)
: i, j는 모두 루트
parent[i]=j



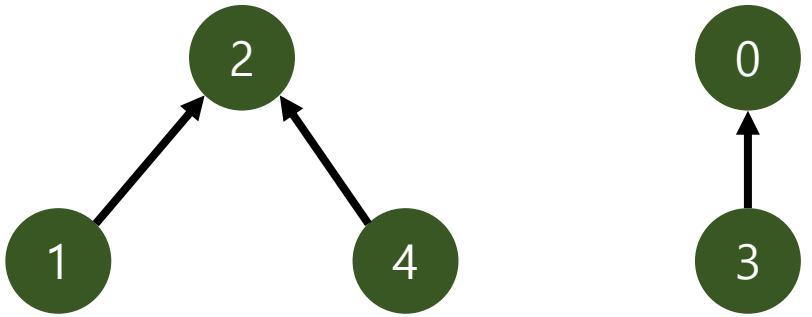
분리 집합 – 성능 향상

i	**0**	1	**2**	3	4
parent	-2	2	-3	0	2

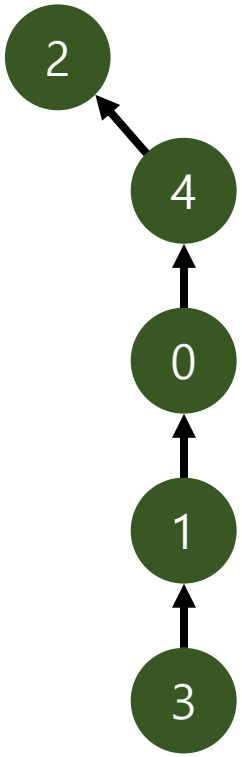
정점의 수 : 2　　정점의 수 : 3

parent[i] < 0 이면 루트
abs(parent[i]) = size[i]

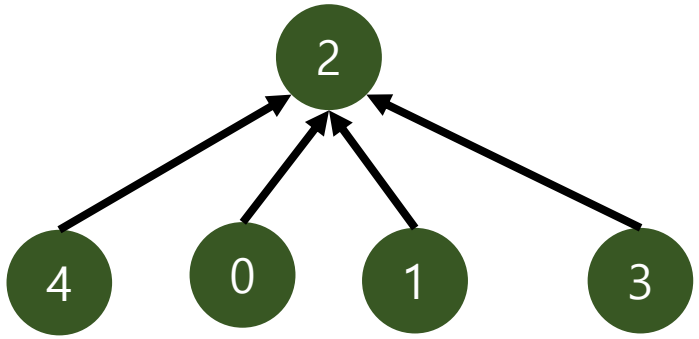
parent[i] >= 0 이면
parent[i]는 정점 i의 부모



분리 집합 : collapsing-find



루트의 높이가 낮아지므로
성능이 좋아진다

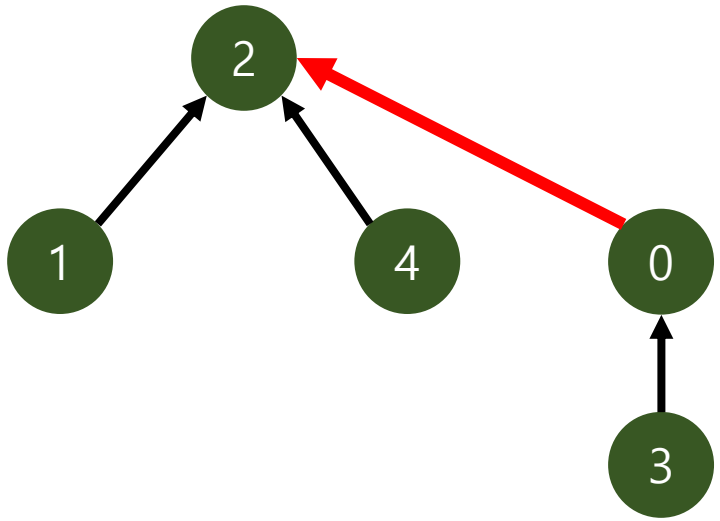


분리 집합 : weighted-union

i	0	1	2	3	4
parent	**2**	2	**-5**	0	2

정점의 개수 : 5

Weighted-union(i, j)
: i, j는 모두 루트
if size[i] < size[j]
then parent[i]=j



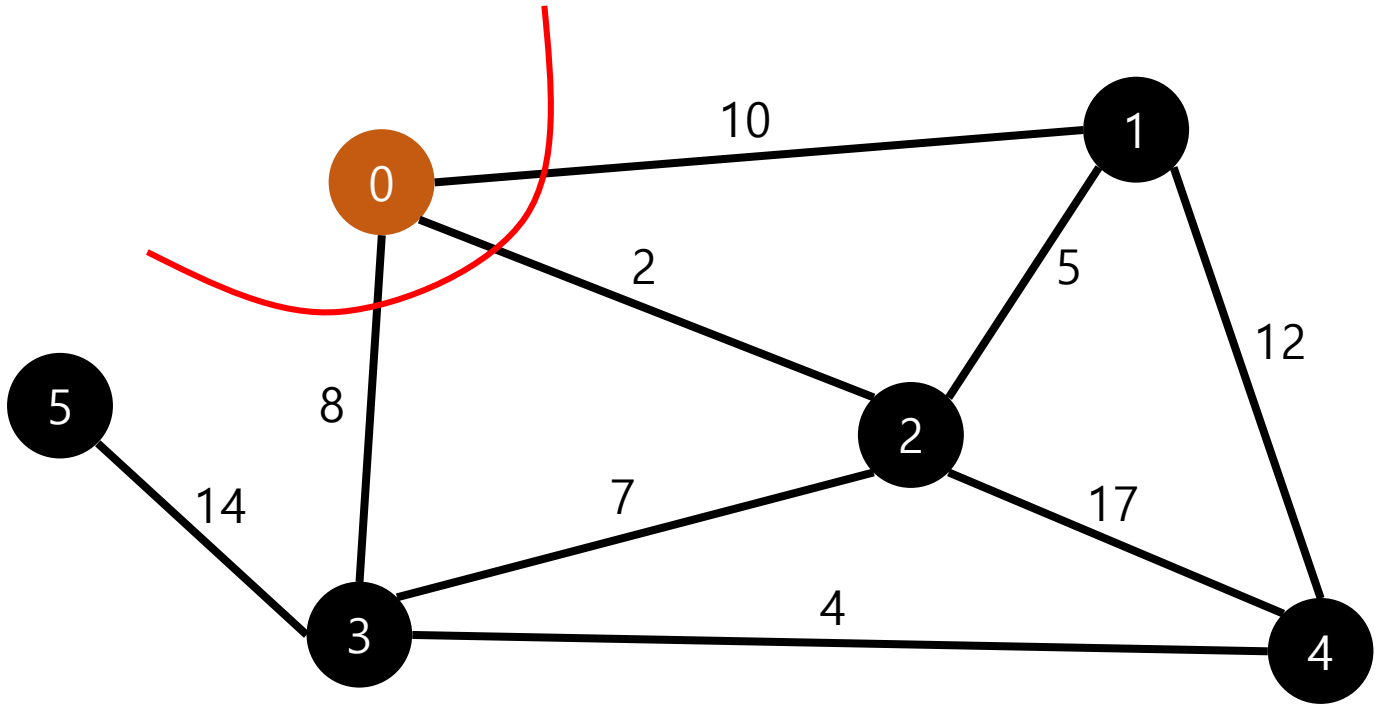
Prim algorithm

Prim algorithm

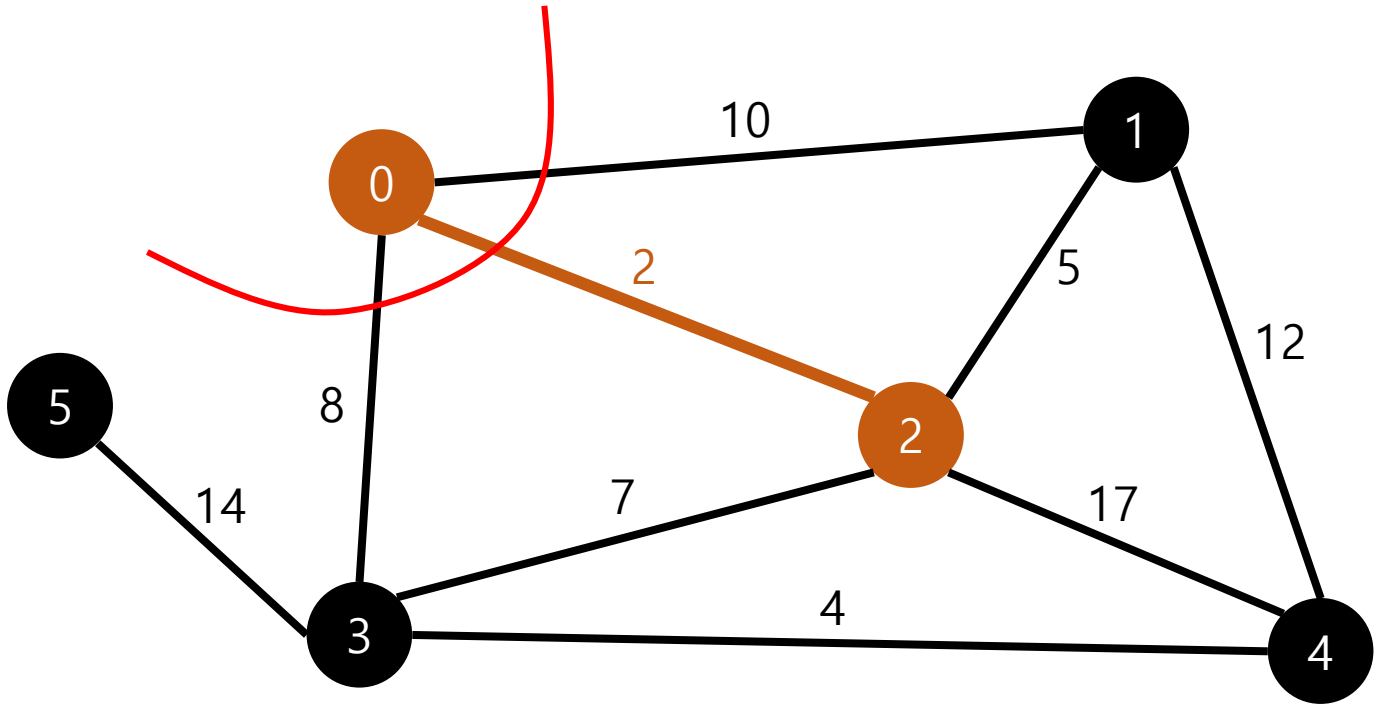
1. 정점 하나를 가진 트리에서 시작
   TV={v1} 여기서 TV는 트리의 정점
2. 트리 내의 정점 u와 트리 밖의 정점 v를 잇는 에지 중
   최소 비용을 가진 (u, v)를 트리 에지로 만든다.
   TE=TE∪{(u, v)}
3. 트리 밖의 정점 v도 트리의 정점으로 만든다.
   TV=TV∪{v}
4. TV=V(G)와 같아지면 종료

트리 내의 정점 u와 트리 밖의 정점 v를 잇는 에지를 선택하므로 사이클은 형성되지 않는다!!

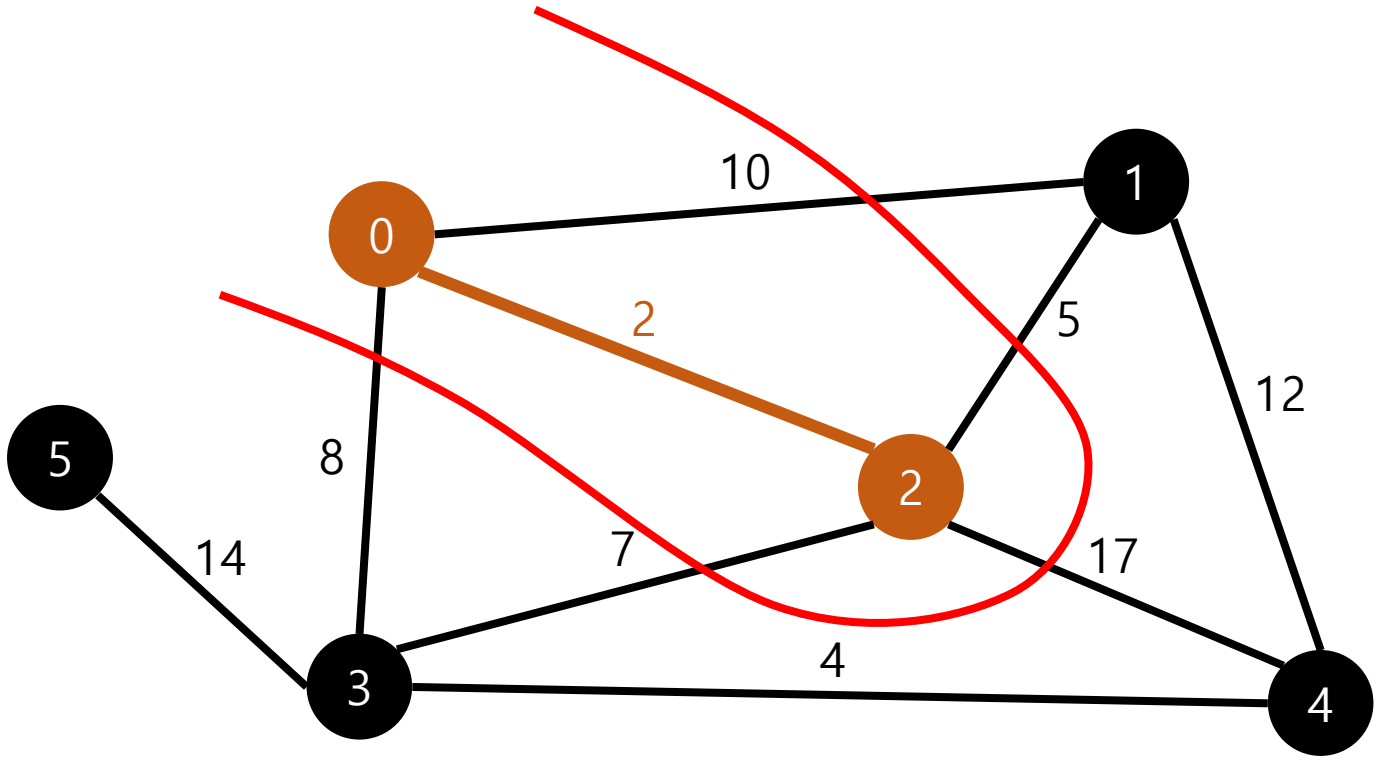
Prim algorithm



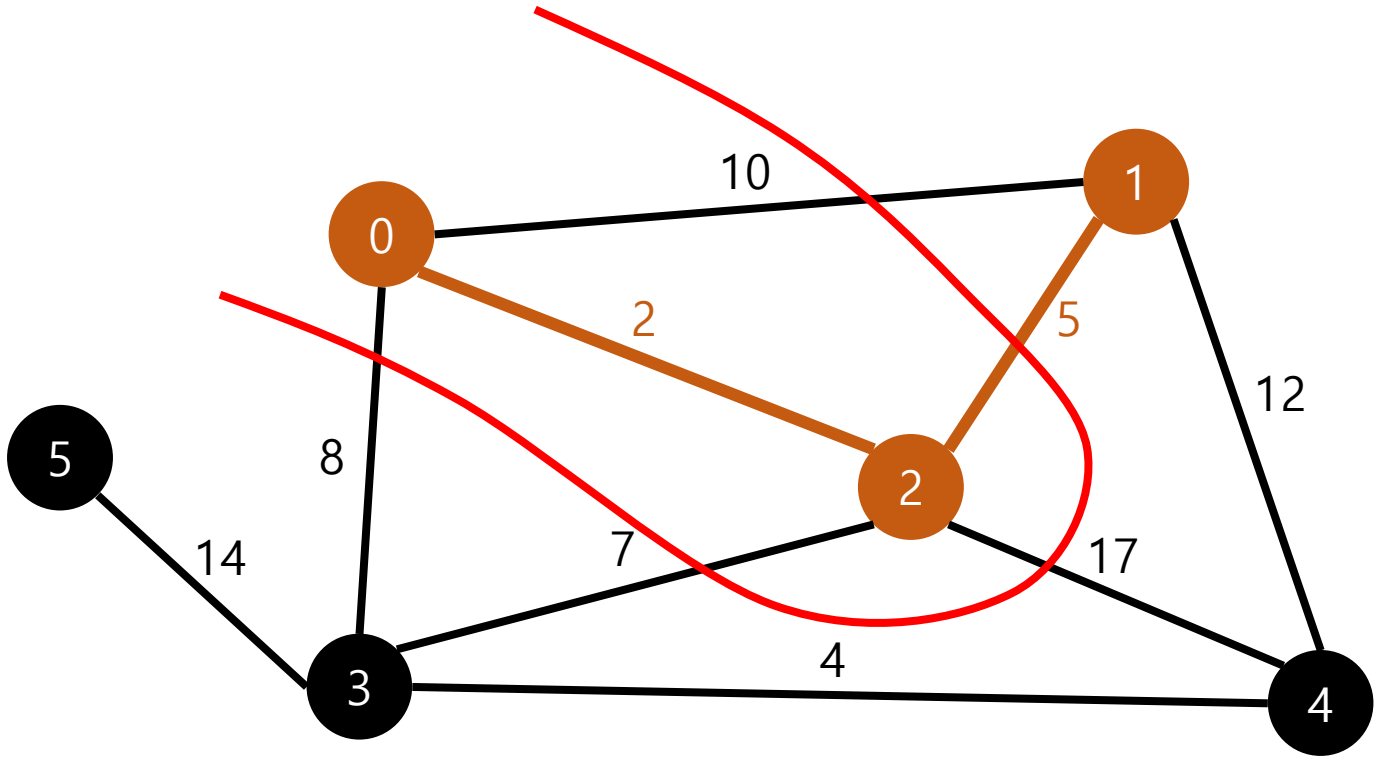
Prim algorithm



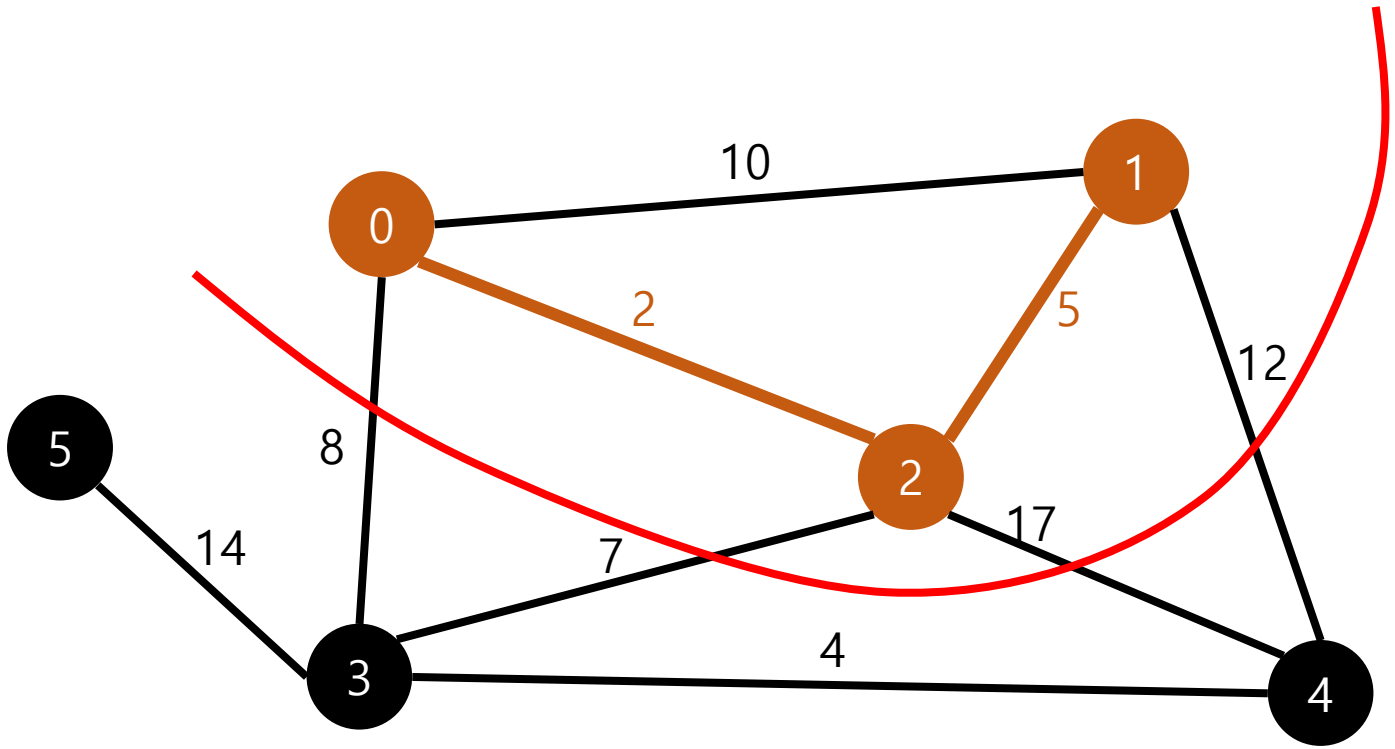
Prim algorithm



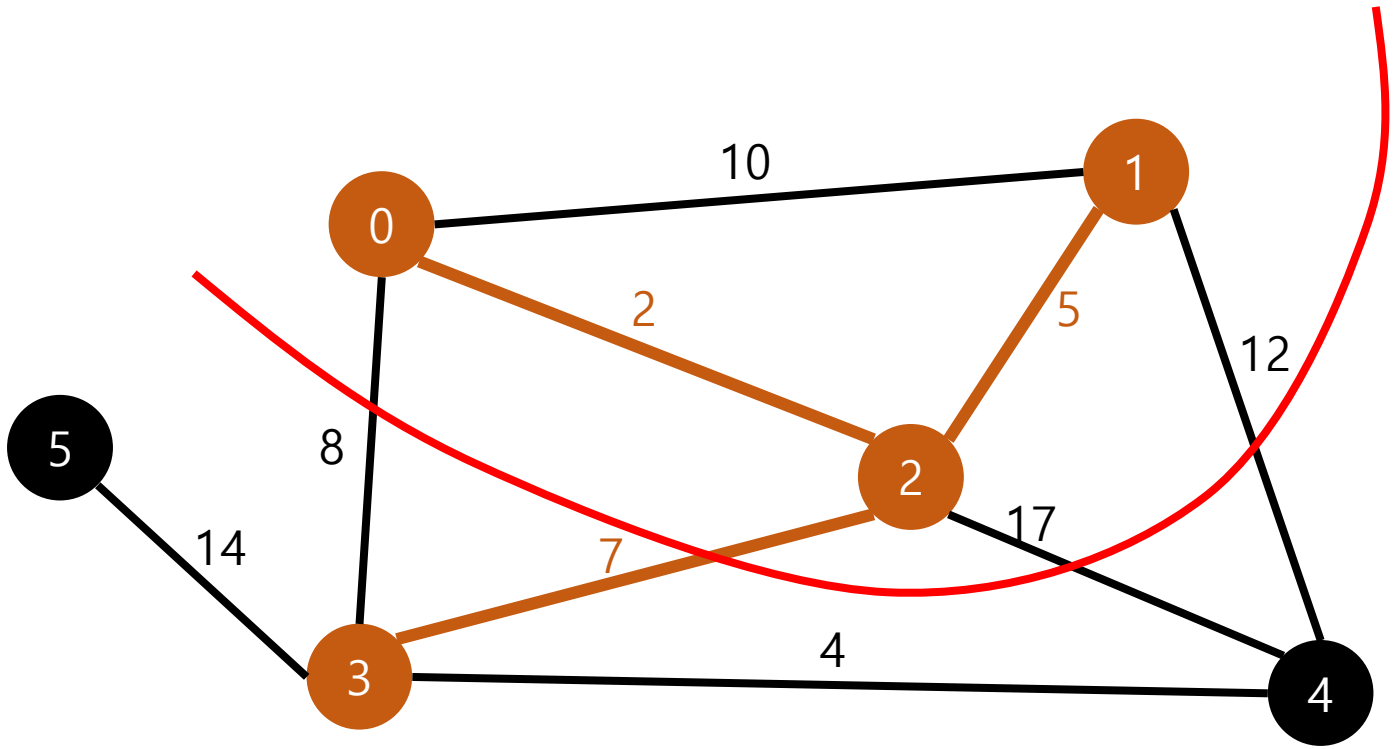
Prim algorithm



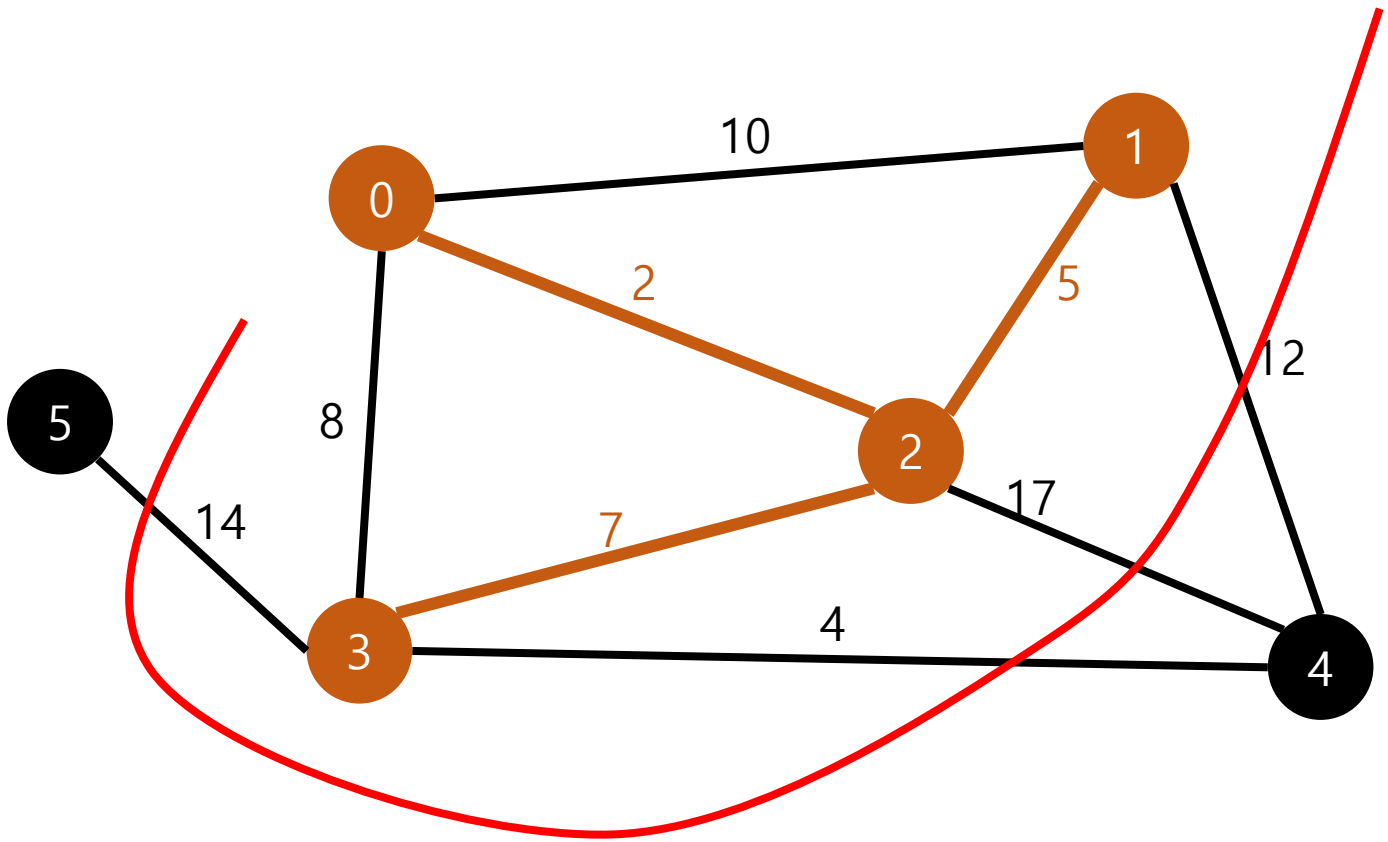
Prim algorithm



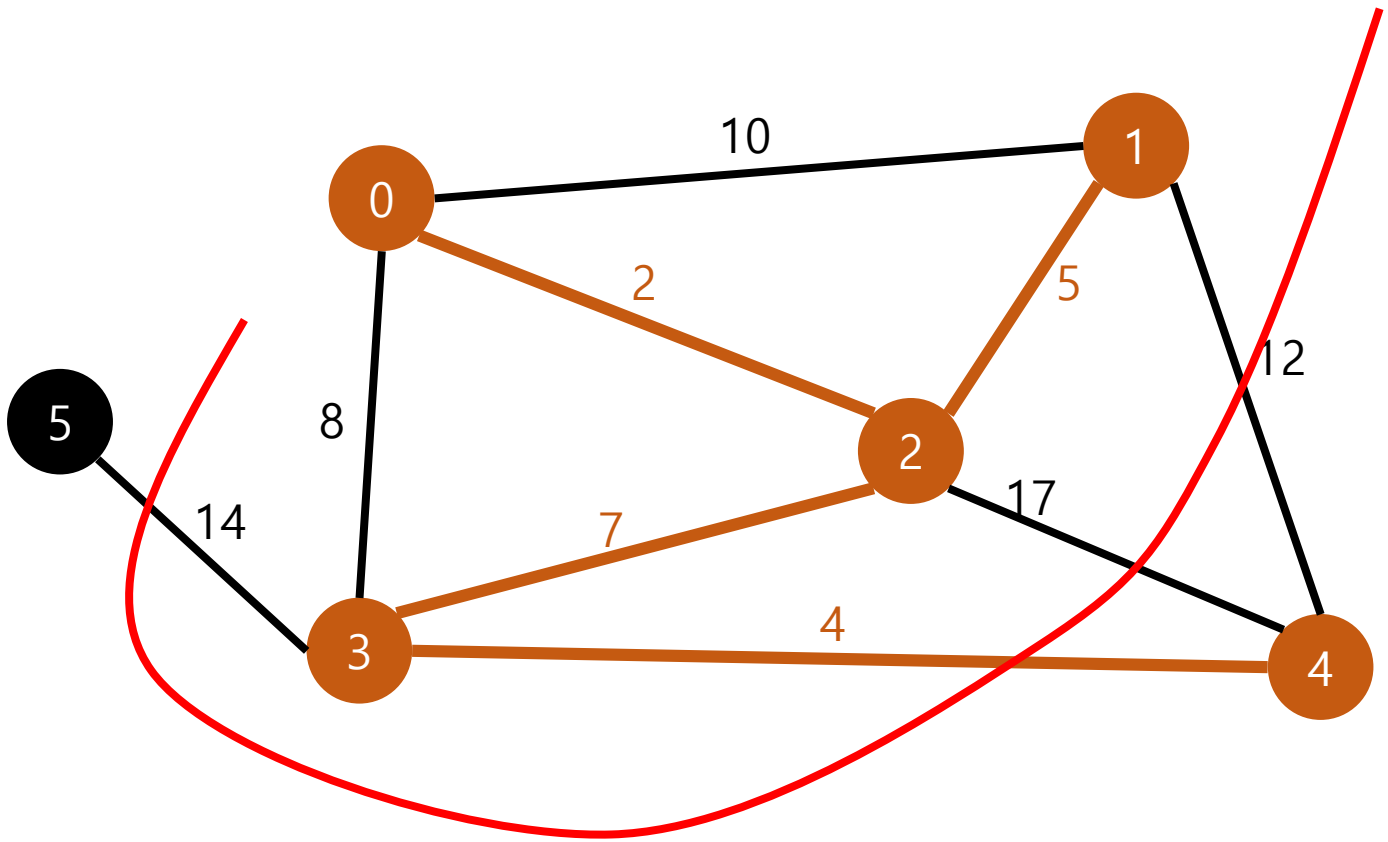
Prim algorithm



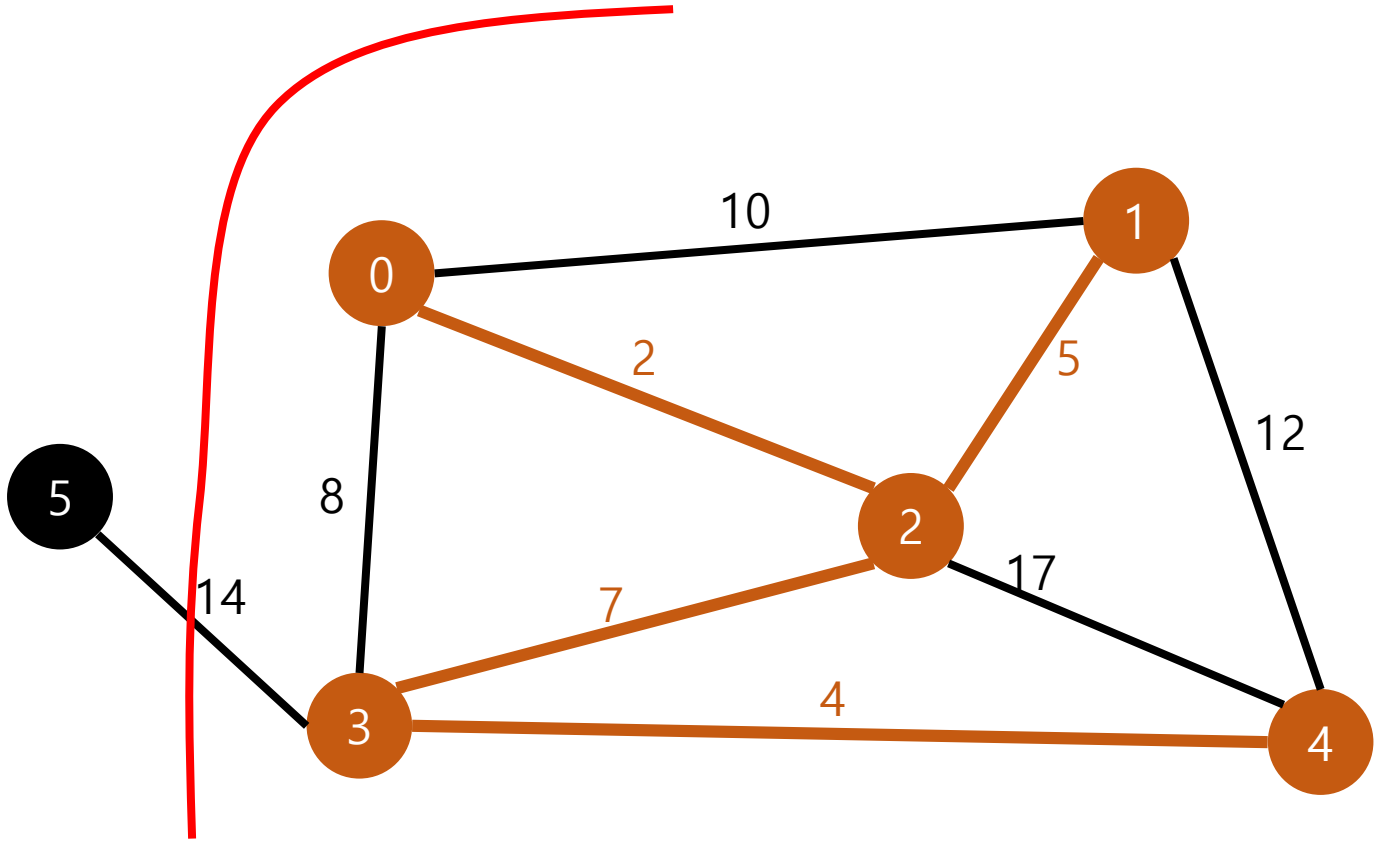
Prim algorithm



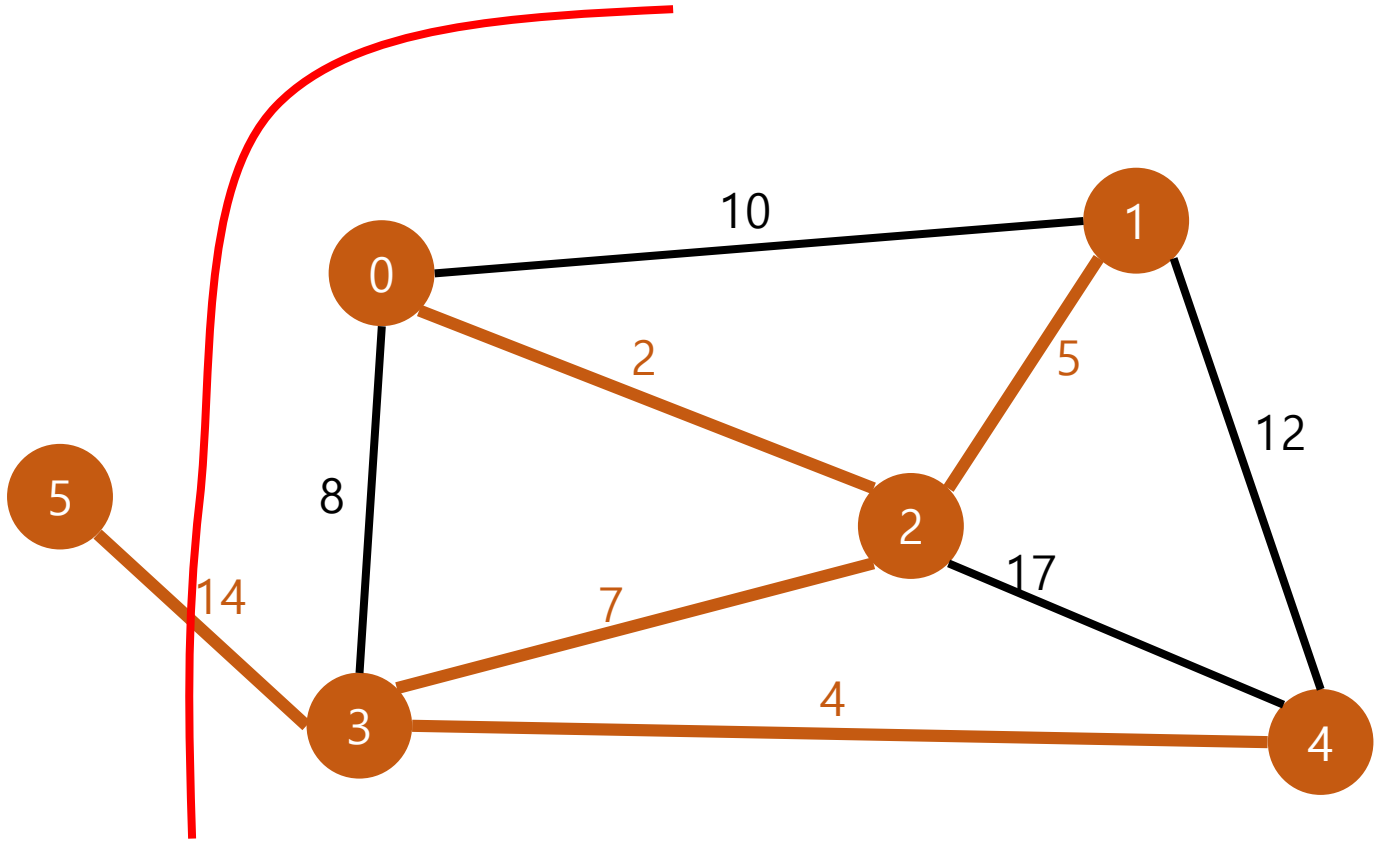
Prim algorithm



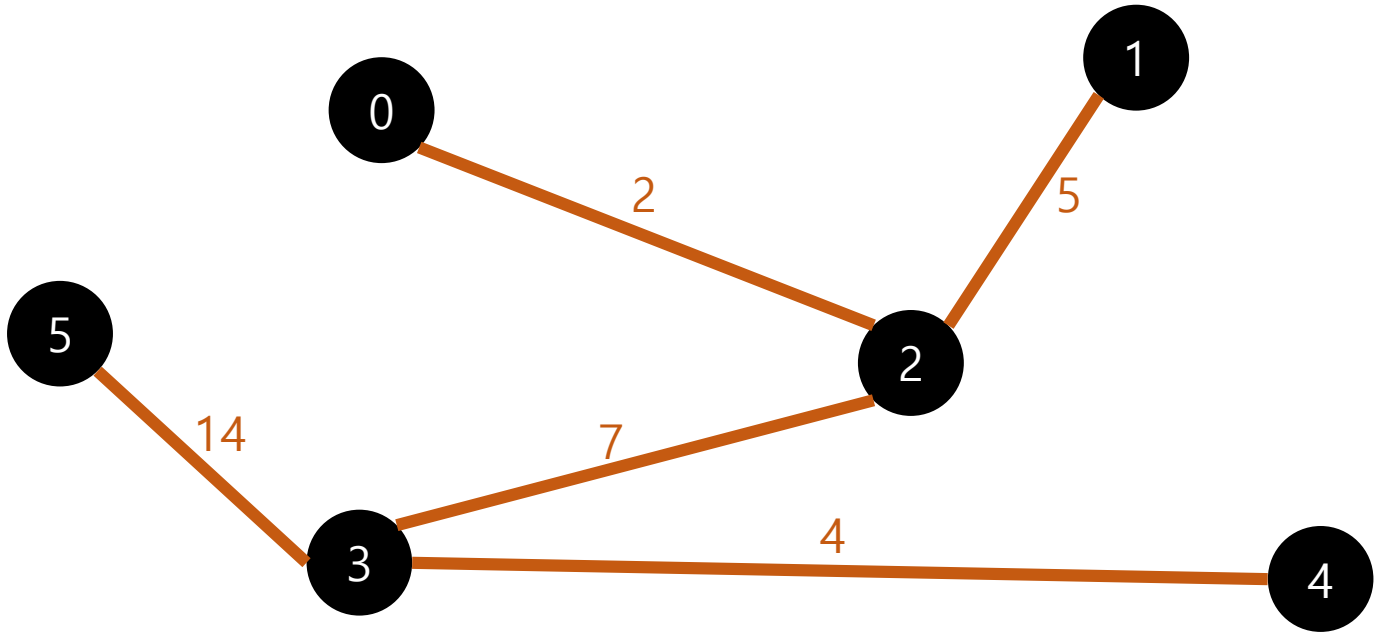
Prim algorithm



Prim algorithm



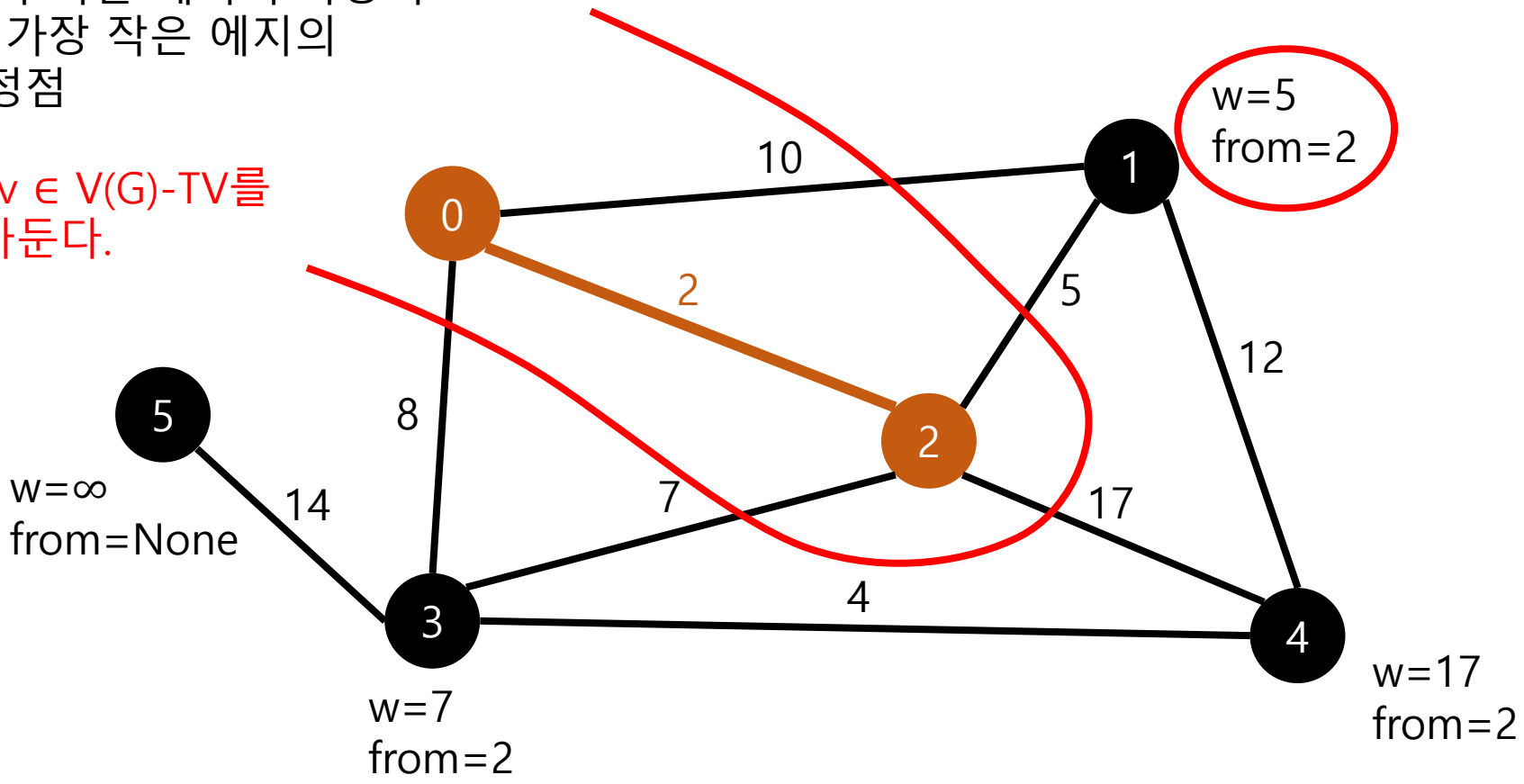
Prim algorithm



최소 비용 에지를 찾는 방법

w : 트리 내 정점과 이어진 에지 중
가중치가 가작 작은 에지의 가중치
from : 가중치가 가장 작은 에지의
트리 내 정점

W와 from을 가진 v ∈ V(G)-TV를
Min heap h에 담아둔다.
힙의 key는 w이다!

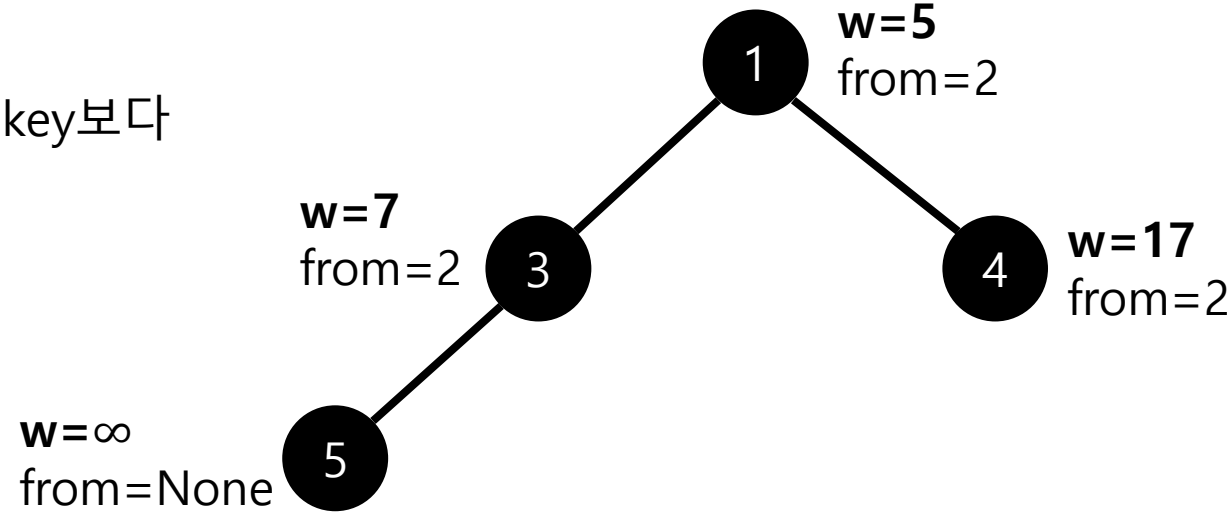


최소 비용 에지를 찾는 방법

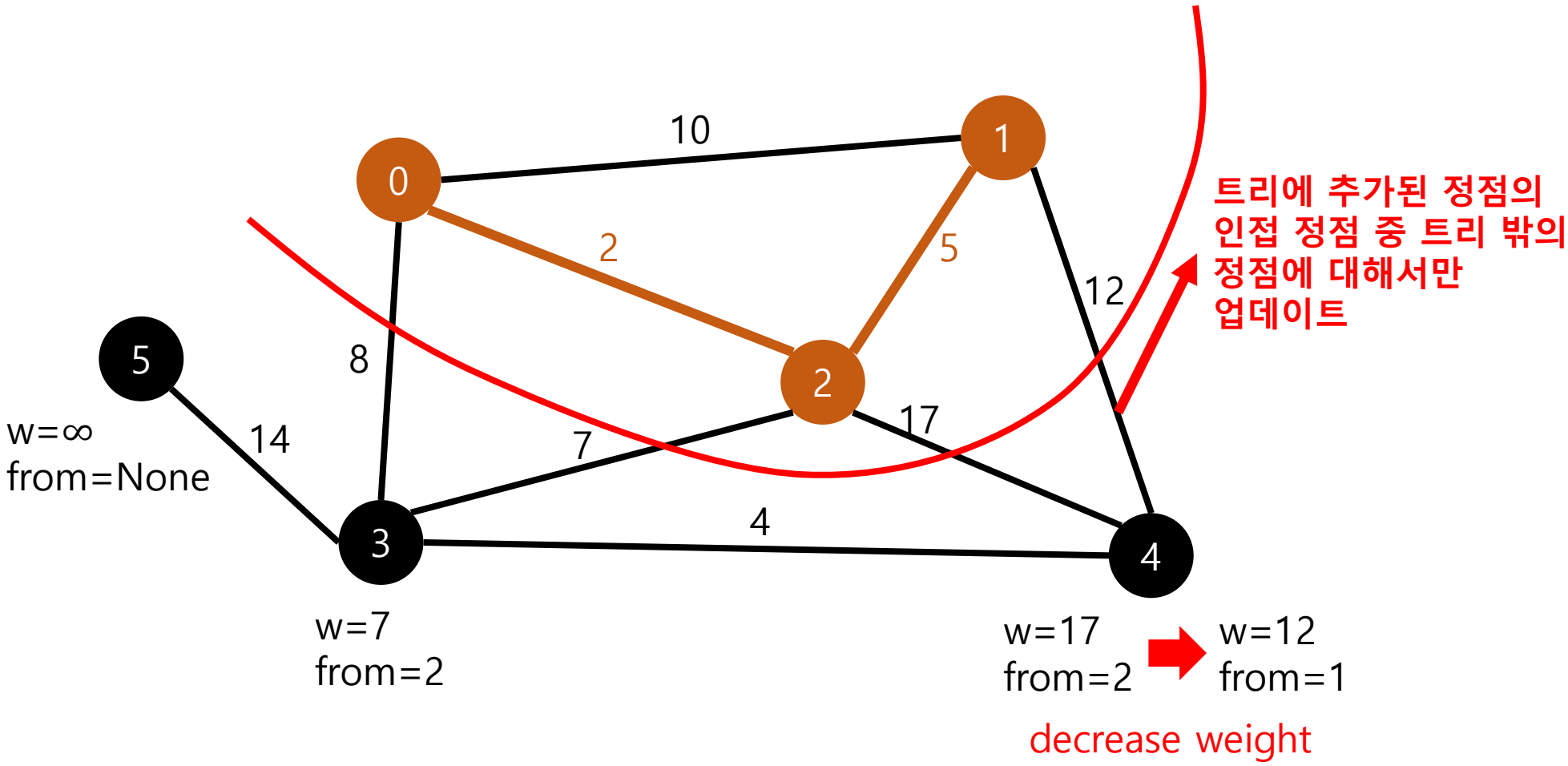
w와 from을 가진 v ∈ V(G)-TV를
Min heap h에 담아둔다.

힙의 key는 w이다!

Heap property
1. 노드의 key는 자식 노드의 key보다 크지 않다.



최소 비용 에지를 찾는 방법



최소 비용 에지를 찾는 방법

Decrease Weight

힙 프로퍼티가 깨지지 않도록
W가 작아진 노드에서 시작해
루트까지 올라가며 재정렬