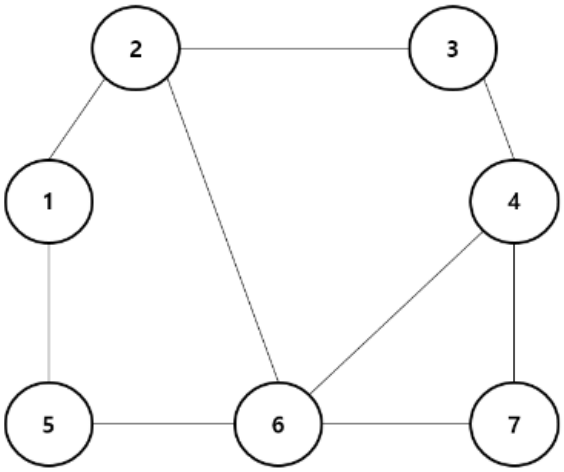
# Kruskal

## Kruskal 알고리즘

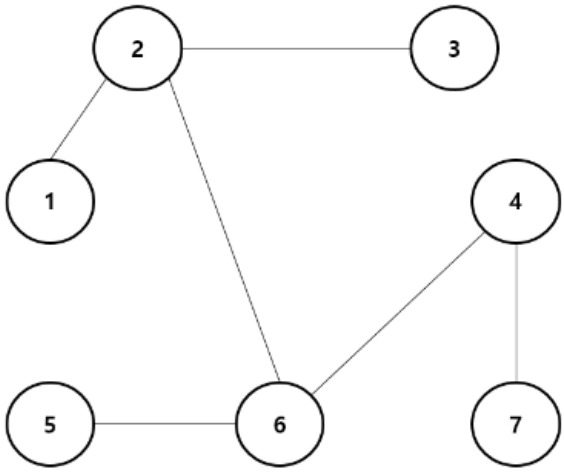
Kruskal 알고리즘은 최소 비용 신장 트리(MST : Minimum Spanning Tree)를 찾는 알고리즘이며 이 알고리즘은 그리디 알고리즘으로 분류된다.

## 신장 트리

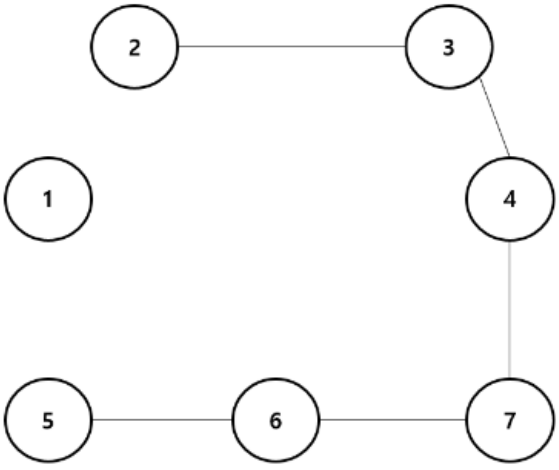
하나의 그래프가 있을 때 모든 노드를 포함하면서 사이클이 존재하지 않는 부분 그래프




원본 그래프

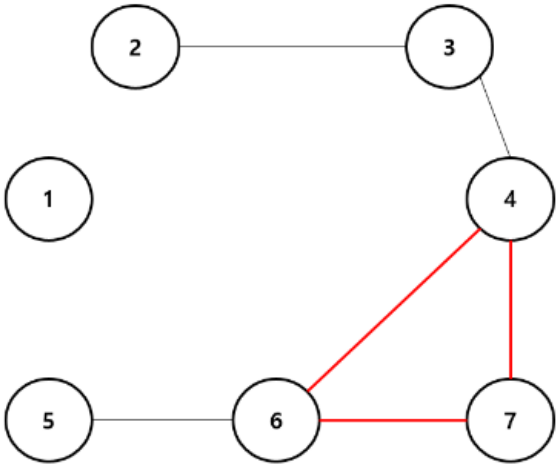



가능한 신장 트리 예시 (O)




신장 트리 아닌 부분 그래프 1 (X)

1번 노드가 연결 X (모든 노드가 연결되어야함)



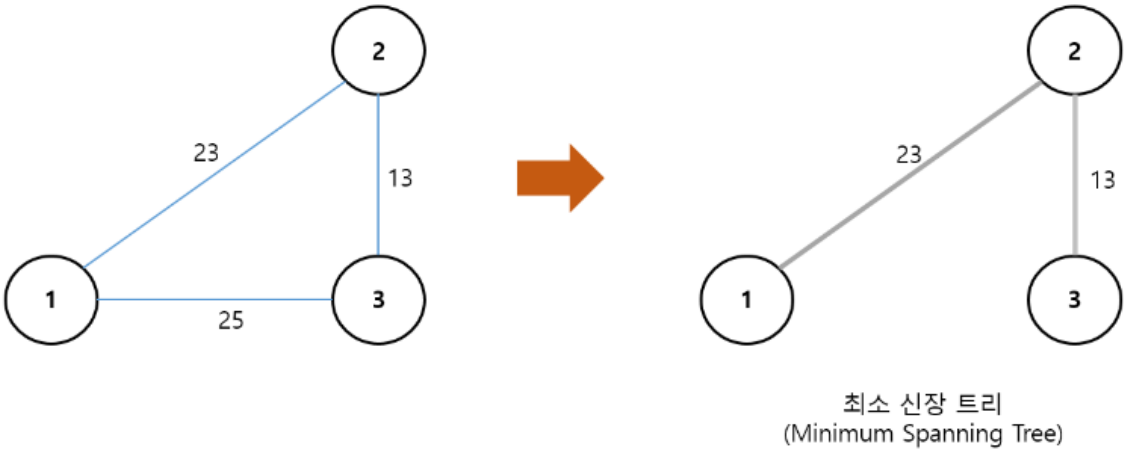

신장 트리 아닌 부분 그래프 2 (X)

4, 6, 7 노드에서 사이클 발생했으므로 신장 트리 X

# 최소 신장 트리(MST : Minimum Spanning Tree)

최소한의 비용으로 만들 수 있는 최소 신장 트리를 찾는 알고리즘

신장 트리의 조건을 만족하면서 최소 비용으로 만들 수 있는 신장 트리가 `최소 신장 트리`




최소 신장 트리
(Minimum Spanning Tree)

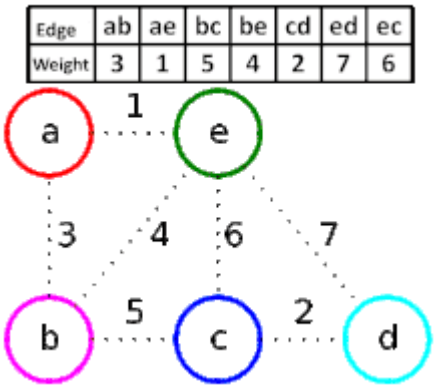
## 알고리즘 동작 방식

1. 그래프의 간선을 가중치가 작은 순서대로 정렬
2. 정렬된 간선들을 하나씩 선택하며 사이클이 형성되지 않는 경우에만 최소 신장 트리에 추가
3. 모든 정점이 최소 신장 트리에 포함될 때까지 2번 과정을 반복

## 시간 복잡도

$T = O(ElogE)$ (where E : 간선의 개수)

## 예시



1. 간선 비용이 오름차순이 되도록 정렬
2. 간선 AE 선택
3. 간선 CD 선택
4. 간선 AB 선택
5. 간선 BE 선택, 사이클이 생기므로 제외
6. 간선 BC 선택
7. 신장 트리 완성

최종적으로 구해진 최소 비용 신장 트리는 AD, AB, CD, D, E의 4개 간선으로 이루어짐

# 구현 예시

union-find 자료구조 활용

```
# 특정 원소가 속한 집합을 찾기
def find(parent, x):
    if parent[x] == x:
        return x
    parent[x] = find(parent, parent[x])
    return parent[x]


# 두 원소가 속한 집합을 합치기 (간선 연결한다고 생각!)
def union(parent, a, b):
    rootA = find(parent, a)
    rootB = find(parent, b)

    if rootA < rootB:
        parent[rootB] = rootA
    else:
        parent[rootA] = rootB


import sys

input = sys.stdin.readline
# 노드의 개수와 간선(union 연산)의 개수 입력받기
v, e = map(int, input().split())
parent = [0] * (v + 1)

edges = []
result = 0

# 부모 테이블상에서, 부모를 자기 자신으로 초기화
for i in range(1, v + 1):
    parent[i] = i

# 모든 간선에 대한 정보를 입력받기
```

```
for _ in range(e):
    a, b, cost = map(int, input().split())
    # 비용순으로 오름차순 정렬하기 위해 튜플의 첫 번째 원소를 비용으로 설정
    edges.append((cost, a, b))

edges.sort()

for edge in edges:
    cost, a, b = edge
    # 사이클이 발생하지 않는 경우에만 집합에 포함(=연결한다.)
    if find(parent, a) != find(parent, b):
        union(parent, a, b)
        result += cost

print(result)

# sample input
# 7 9
# 1 2 29
# 1 6 75
# 2 3 35
# 2 6 34
# 3 4 7
# 4 6 23
# 4 7 13
# 5 6 53
# 6 7 25
```