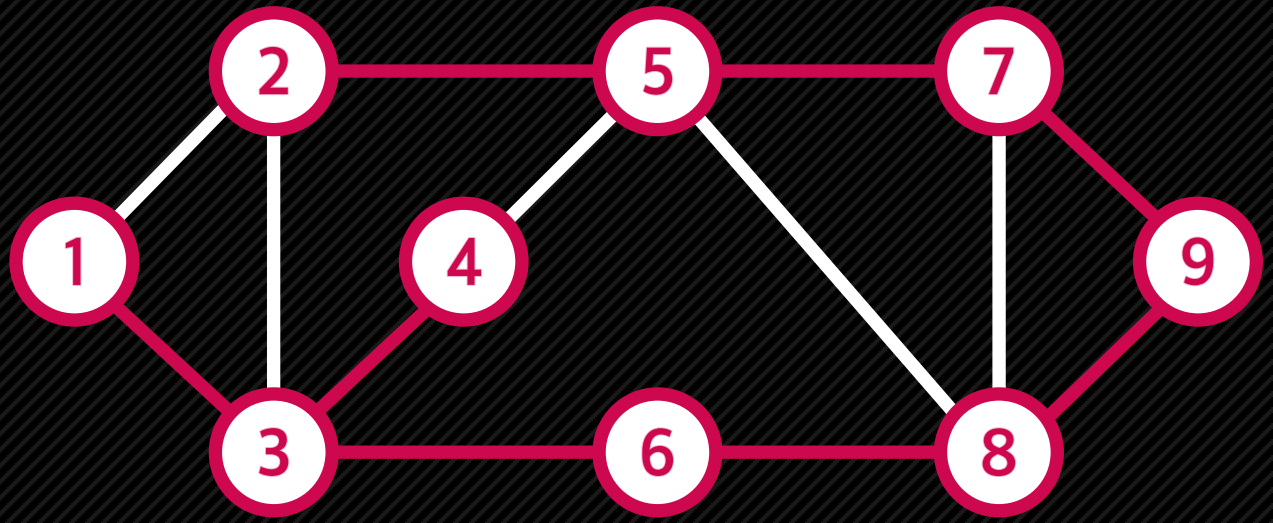
알고리즘 특강
그래프 알고리즘

다양한 그래프 알고리즘들에 대해 알아봅시다.
최단거리 및 최소 신장 트리 알고리즘과, 활용법에 대해 배웁니다.

지난 시간에…


특정 정점에서의 거리
모든 정점과 정점의 거리
정점에 가중치 없음
BFS
Floyd-Warshell Algorithm
정점에 가중치 있음
음수 가중치 있음
Bellman-Ford Algorithm
음수 가중치 없음
Dijkstra Algorithm

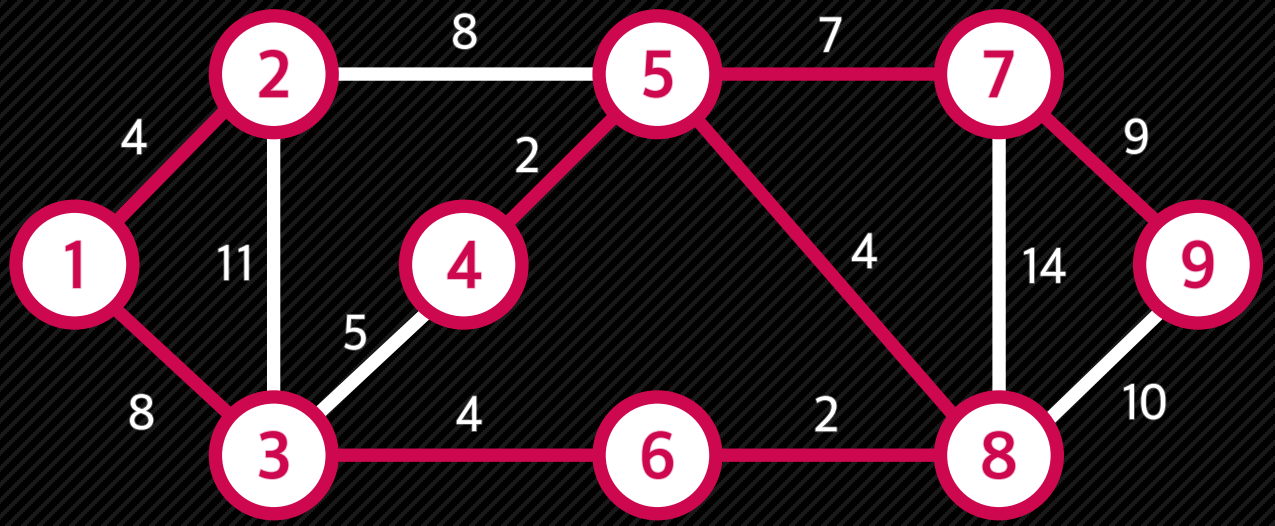
MST



Spanning Tree

- 그래프의 최소한의 간선만을 채택하여, 모든 정점이 이어지도록 만든 그래프.
- N개의 정점을 이어지게 하기 위해선 정확히 (N - 1)개의 간선이 필요하다.
- N개의 정점과 (N - 1)개의 간선으로 이루어진 형태이기 때문에, 트리라고 부를 수 있다.

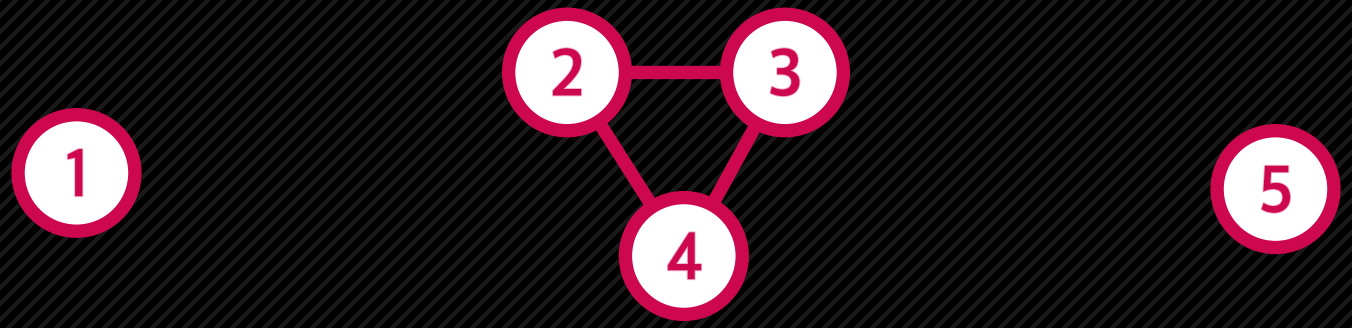
MST



Minimum Spanning Tree

- Spanning Tree 중에서 가중치의 합이 가장 작은 Spanning Tree.
- MST를 구하는 대표적인 알고리즘은 Kruskal Algorithm, Prim Algorithm이 있다.

Union-Find



Union-Find

- 서로 중복되지 않는 부분 집합들로 (**상호 배타적**!) 나눠진 **원소들에 대한 정보**를 관리하는 자료구조
- 특정 원소가 **어떤 집합에 속해 있는지** 확인하는 메소드와, **두 집합을 합치는** 메소드로 이루어져 있다.

Union-Find

- 특정 원소가 어떤 집합에 속해 있는지 확인하는 메소드와, 두 집합을 합치는 메소드로 이루어져 있다.

`find()` `union()`

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

Union-Find

union()

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

Union-Find

union()

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

Union-Find

union()

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

Union-Find

union()

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

Union-Find

union()

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

Union-Find

union()

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

Union-Find

find()

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

0이 나올때 까지 역추적 하면 되지 않을까?

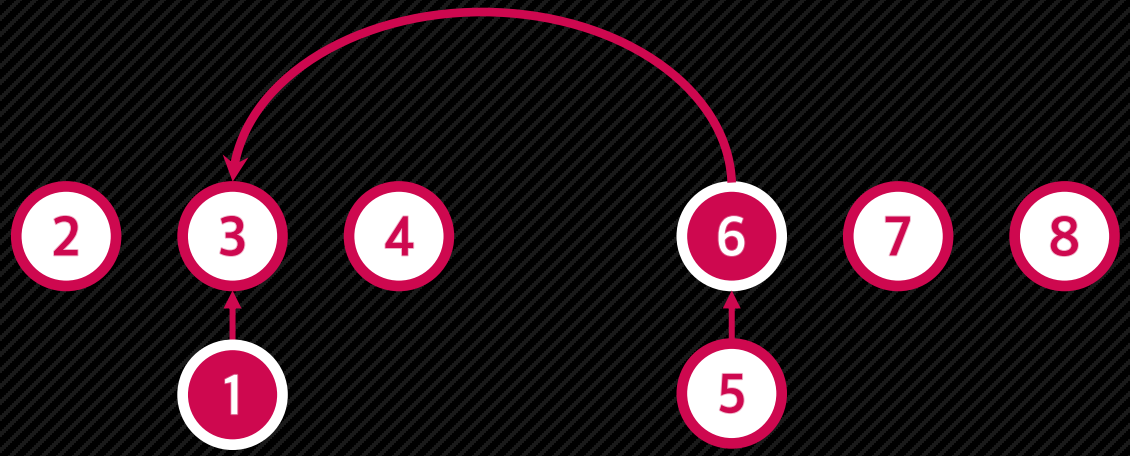
그런데…

find()

저 세상으로 가는 시간 복잡도…

이렇게 할 순 없을까?

union()



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

이렇게 할 순 없을까?

```
function find(x):
    if uf[x] == -1
        return x
    return uf[x] <- find(uf[x])
```

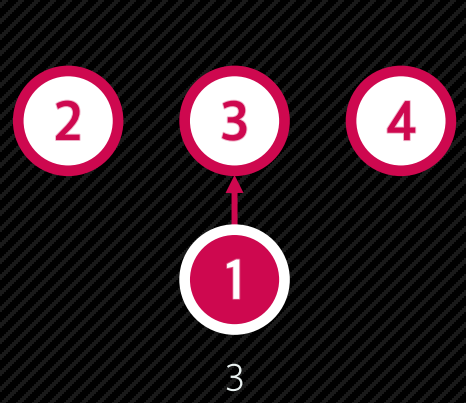
```
function union(x, y):
    set X <- find(x), Y <- find(y)
    uf[X] <- Y
```

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

이렇게 할 순 없을까?

```
function find(x):
    if uf[x] == -1
        return x
    return uf[x] <- find(uf[x])
```

```
function union(x, y):
    set X <- find(x), Y <- find(y)
    uf[X] <- Y
```



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

이렇게 할 순 없을까?

```
function find(x):
    if uf[x] == -1
        return x
    return uf[x] <- find(uf[x])
```

```
function union(x, y):
    set X <- find(x), Y <- find(y)
    uf[X] <- Y
```

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

이렇게 할 순 없을까?

```
function find(x):
    if uf[x] == -1
        return x
    return uf[x] <- find(uf[x])
```

```
function union(x, y):
    set X <- find(x), Y <- find(y)
    uf[X] <- Y
```

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

한번 굴려 봅시다.

Gold 4 - 집합의 표현 (#1717)

요약

- 초기에 {0}, {1}, {2}, …, {n}이 각각 n + 1개의 집합을 구성하고 있다.
- 여기에 합집합 연산과 두 원소가 같은 집합에 포함되어 있는지 확인하는 연산을 수행하려고 한다.
- 집합을 표현하는 프로그램을 작성하시오.

제약조건

- N의 범위는 1 <= N <= 1,000,000 이다.
- 연산의 수 M의 범위는 1 <= M <= 100,000 이다.

직접 해볼까요?

Gold 4 - 사이클 게임 (#20040)

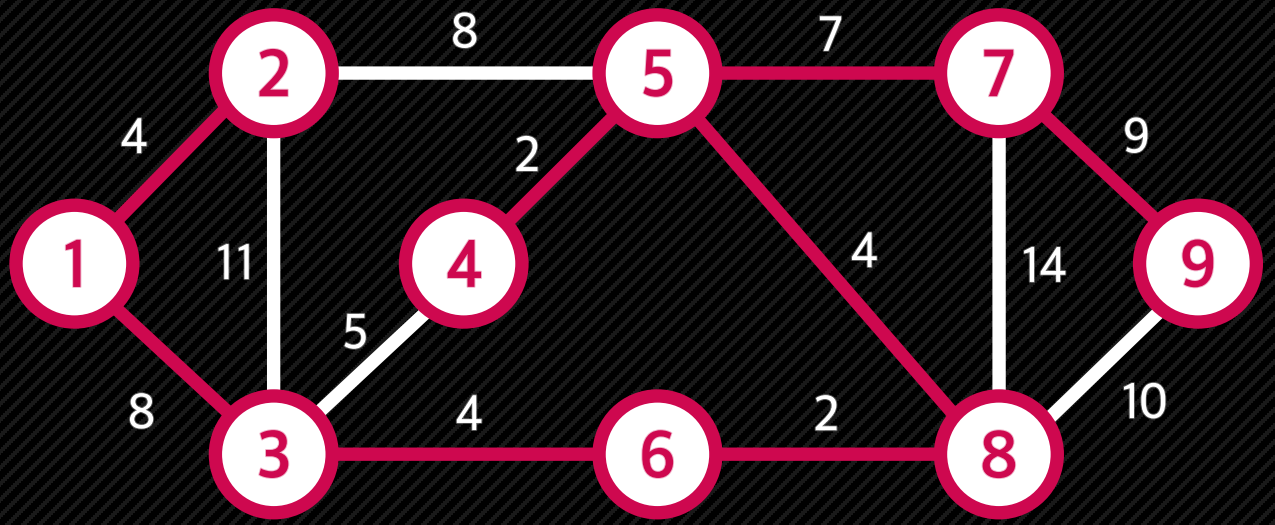
요약

- 0부터 n - 1까지 번호가 매겨진 평면 상의 점 n개가 주어지는데, 어떠한 세 점도 한 직선 위에 있지 않다.
- 매 차례 플레이어는 돌아가면서 두 점을 잇는데, 이때 사이클이 나오면 게임이 종료된다.
- 게임의 어느 시점에 사이클이 발생하였는지 확인하는 프로그램을 작성하시오.

제약조건

- N의 범위는 3 <= N <= 500,000 이다.
- 차례의 수 M의 범위는 3 <= M <= 1,000,000 이다.

다시 MST로…



Minimum Spanning Tree

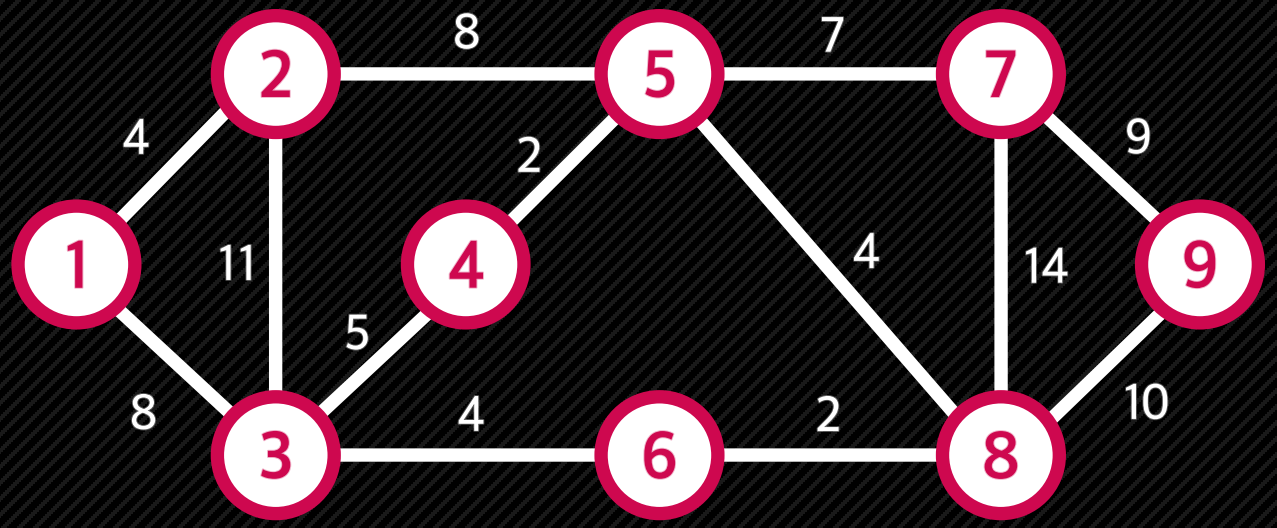
- Spanning Tree 중에서 가중치의 합이 가장 작은 Spanning Tree.
- MST를 구하는 대표적인 알고리즘은 Kruskal Algorithm, Prim Algorithm이 있다.

Kruskal?

Kruskal Algorithm

- 가중치가 낮은 간선부터 탐욕적으로 선택해 MST를 구성하는 알고리즘.

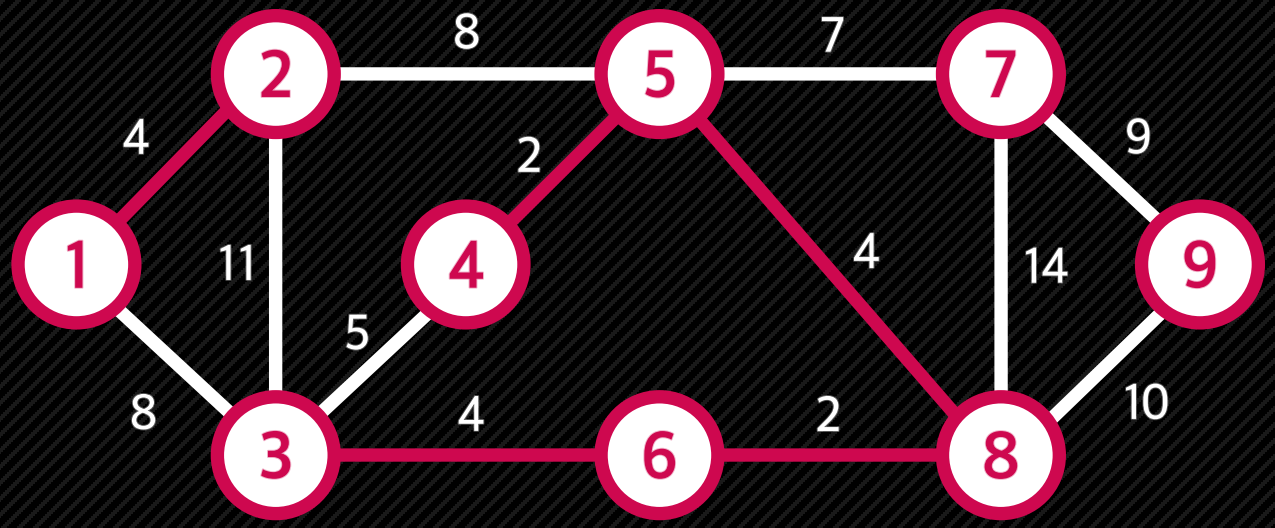
일단.. 해보자.



Kruskal Algorithm

- 가중치가 낮은 간선부터 탐욕적으로 선택해 MST를 구성하는 알고리즘.

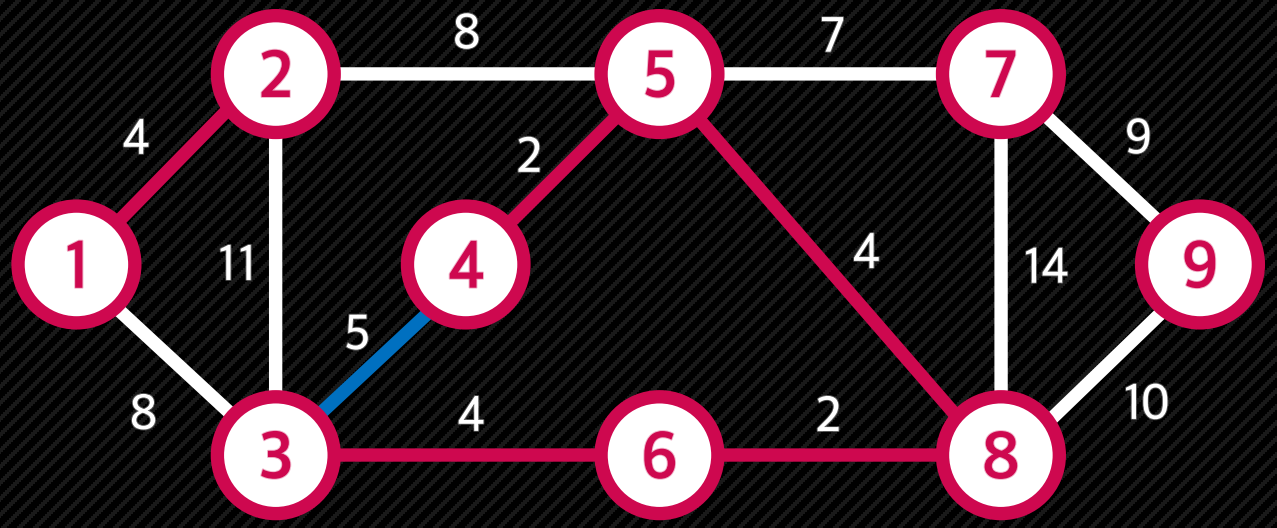
일단.. 해보자.



Kruskal Algorithm

- 가중치가 낮은 간선부터 탐욕적으로 선택해 MST를 구성하는 알고리즘.

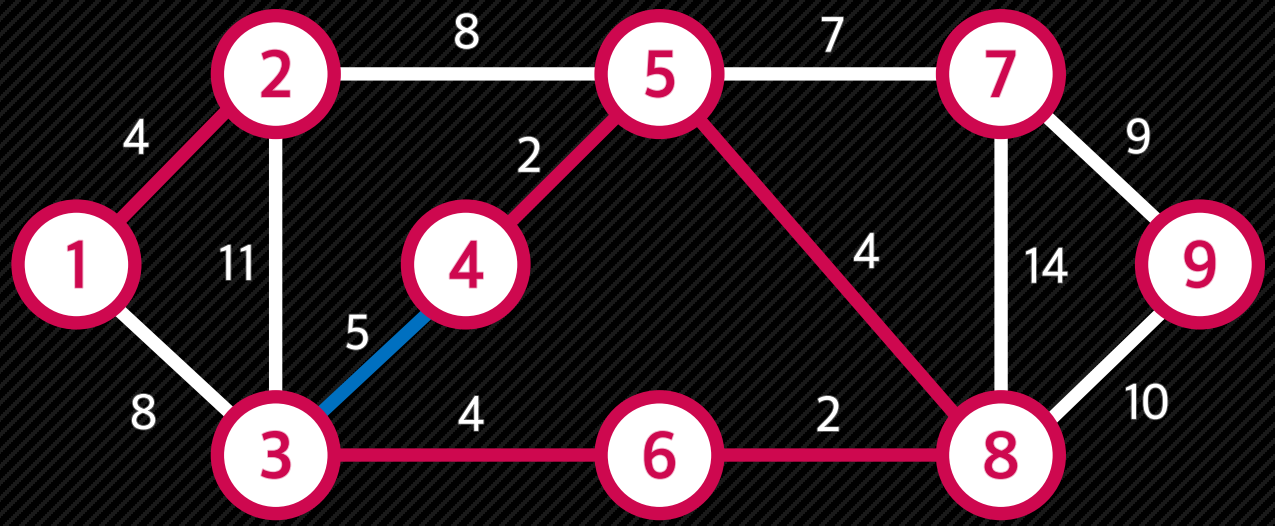
어?



Spanning Tree

- 그래프의 최소한의 간선만을 채택하여, 모든 정점이 이어지도록 만든 그래프.
- N개의 정점을 이어지게 하기 위해선 정확히 (N - 1)개의 간선이 필요하다.

Union-Find



Union-Find

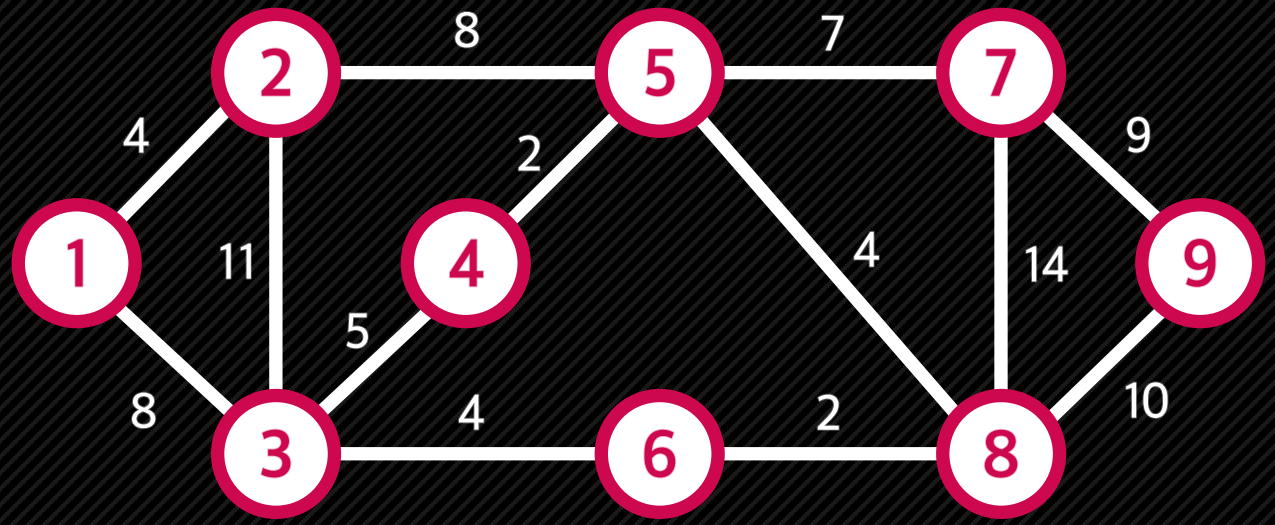
- 서로 중복되지 않는 부분 집합들로 (**상호 배타적**!) 나눠진 **원소들에 대한 정보**를 관리하는 자료구조
- 특정 원소가 **어떤 집합에 속해 있는지** 확인하는 메소드와, **두 집합을 합치는** 메소드로 이루어져 있다.

→ 간선을 선택할 때, 같은 집합 내에 있으면 선택하지 말고 넘어가는 건 어떨까?

Kruskal

```
function kruskal():
    set mst <- |V| - 1 size of array
    sort edge[] by length
    set union-find initialized to -1
    for each E in edge[]
        if find(u) != find(v)
        push E to mst
        union(u, v)
    return mst
```

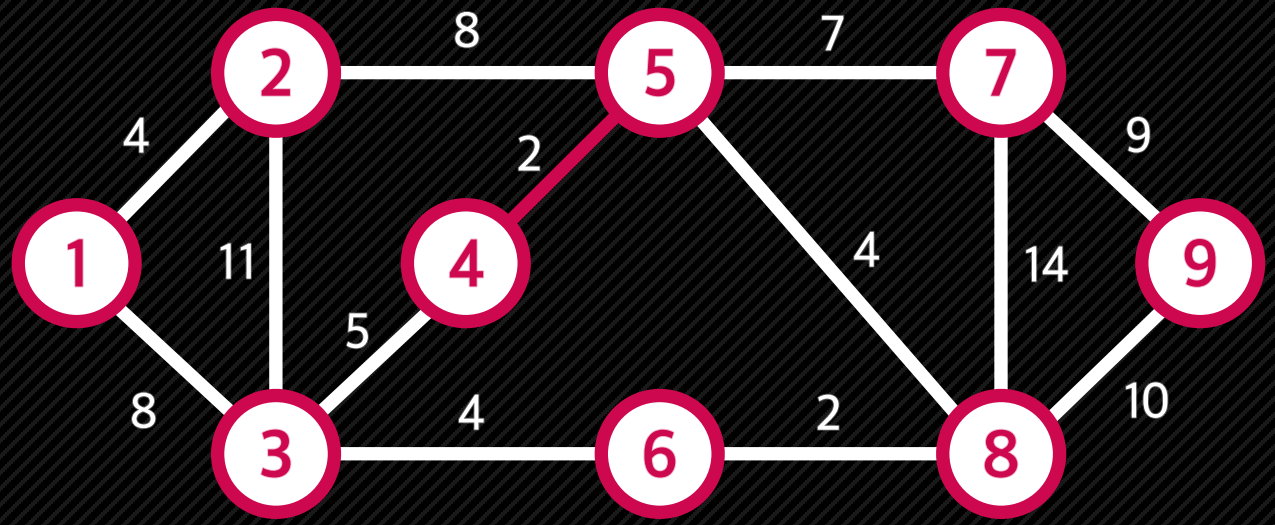
Kruskal



	1	2	3	4	5	6	7	8	9
uf	-1	-1	-1	5	-1	-1	-1	-1	-1

```
function kruskal():
    set mst <- |V| - 1 size of array
    sort edge[] by length
    set union-find initialized to -1
    for each E in edge[]
        if find(u) != find(v)
        push E to mst
        union(u, v)
    return mst
```

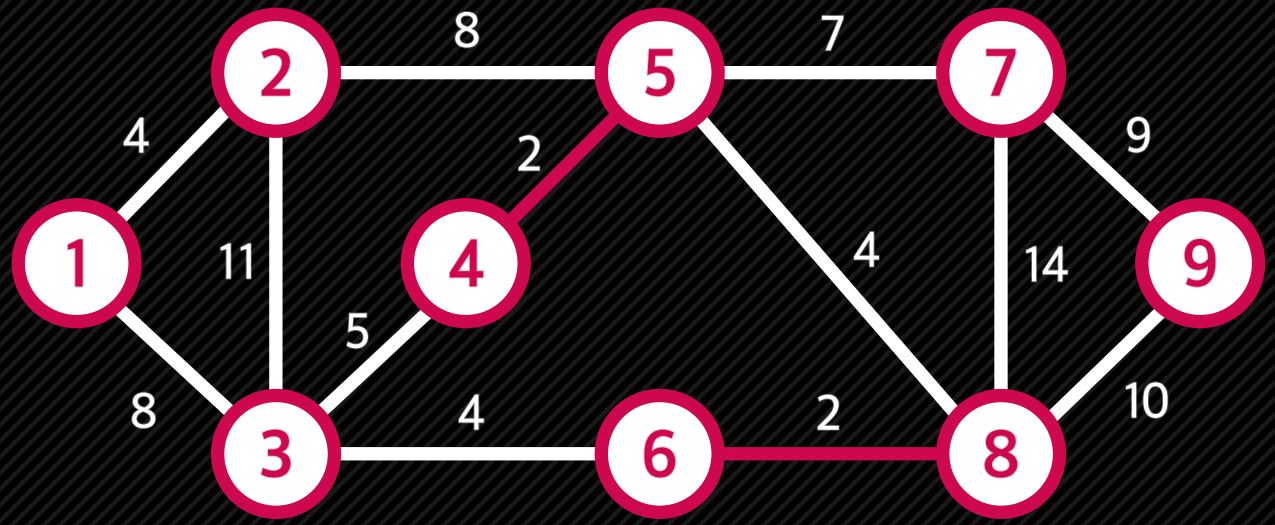
Kruskal



	1	2	3	4	5	6	7	8	9
uf	-1	-1	-1	5	-1	-1	-1	-1	-1

```
function kruskal():
    set mst <- |V| - 1 size of array
    sort edge[] by length
    set union-find initialized to -1
    for each E in edge[]
        if find(u) != find(v)
        push E to mst
        union(u, v)
    return mst
```

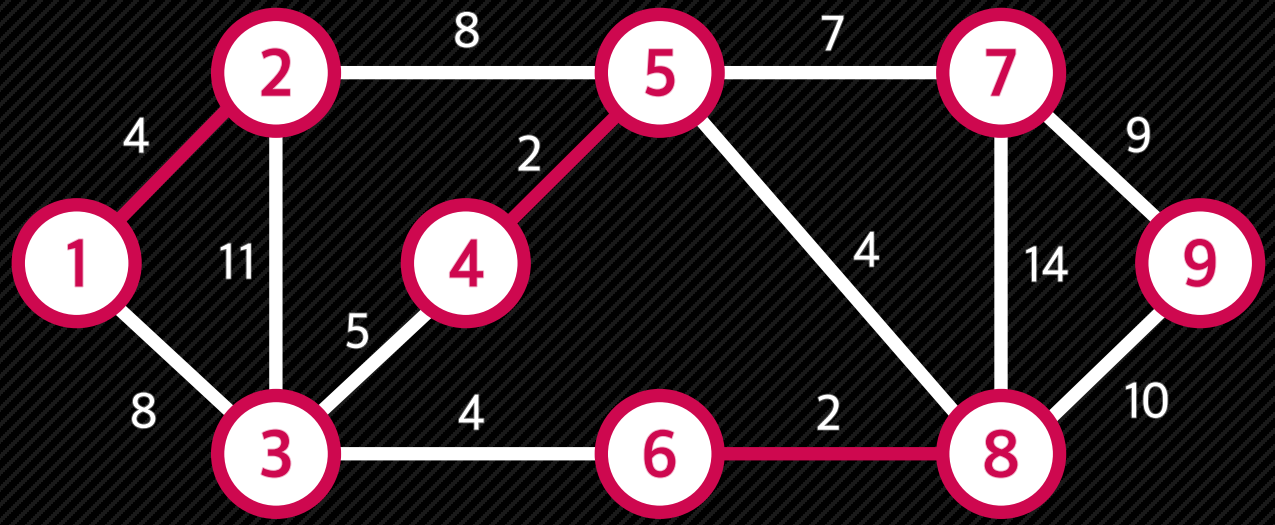
Kruskal



	1	2	3	4	5	6	7	8	9
uf	-1	-1	-1	5	-1	8	-1	-1	-1

```
function kruskal():
    set mst <- |V| - 1 size of array
    sort edge[] by length
    set union-find initialized to -1
    for each E in edge[]
        if find(u) != find(v)
        push E to mst
        union(u, v)
    return mst
```

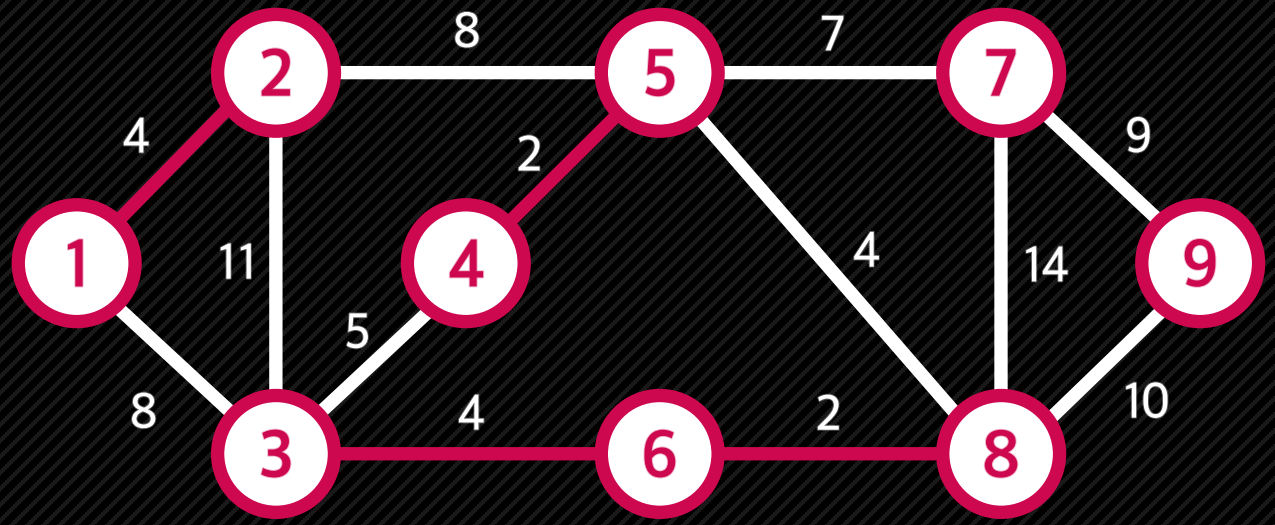
Kruskal



	1	2	3	4	5	6	7	8	9
uf	2	-1	-1	5	-1	8	-1	-1	-1

```
function kruskal():
    set mst <- |V| - 1 size of array
    sort edge[] by length
    set union-find initialized to -1
    for each E in edge[]
        if find(u) != find(v)
        push E to mst
        union(u, v)
    return mst
```

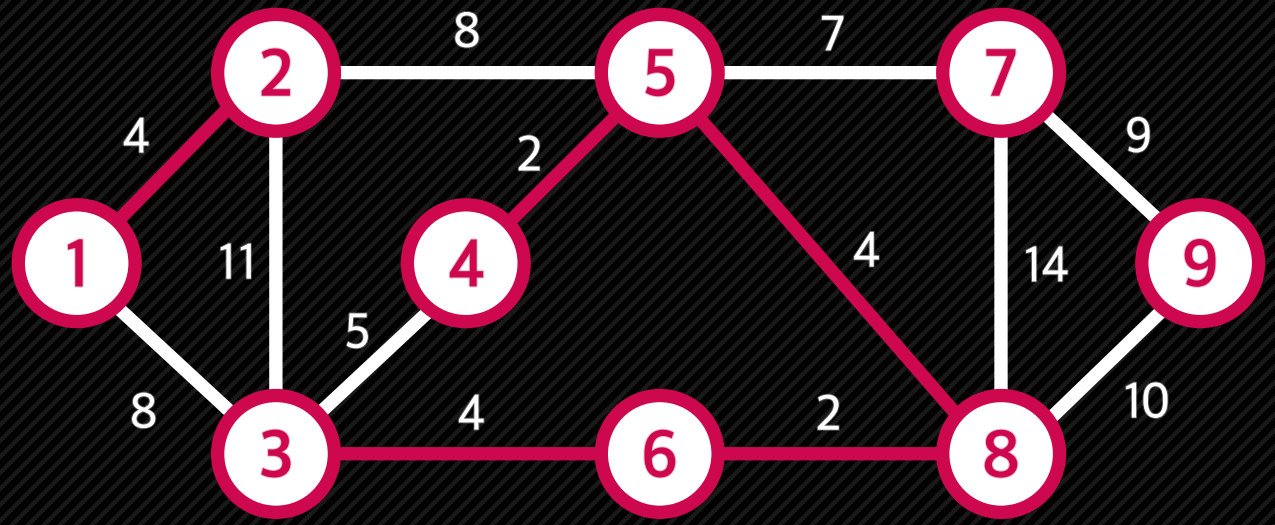
Kruskal



	1	2	3	4	5	6	7	8	9
uf	2	-1	8	5	-1	8	-1	-1	-1

```
function kruskal():
    set mst <- |V| - 1 size of array
    sort edge[] by length
    set union-find initialized to -1
    for each E in edge[]
        if find(u) != find(v)
        push E to mst
        union(u, v)
    return mst
```

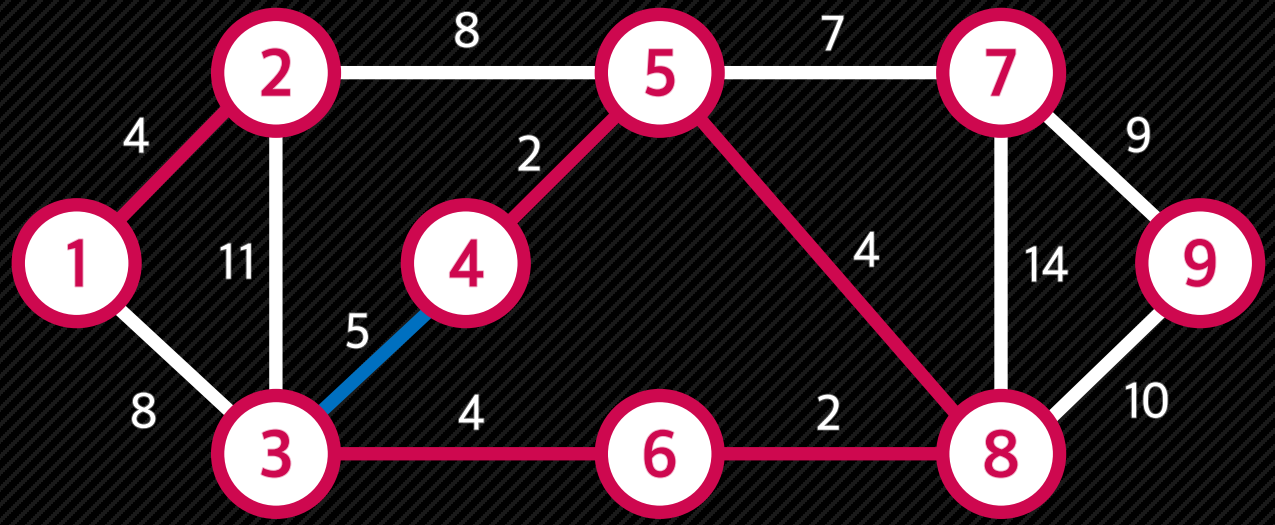
Kruskal



	1	2	3	4	5	6	7	8	9
uf	2	-1	8	5	8	8	-1	-1	-1

```
function kruskal():
    set mst <- |V| - 1 size of array
    sort edge[] by length
    set union-find initialized to -1
    for each E in edge[]
        if find(u) != find(v)
        push E to mst
        union(u, v)
    return mst
```

Kruskal

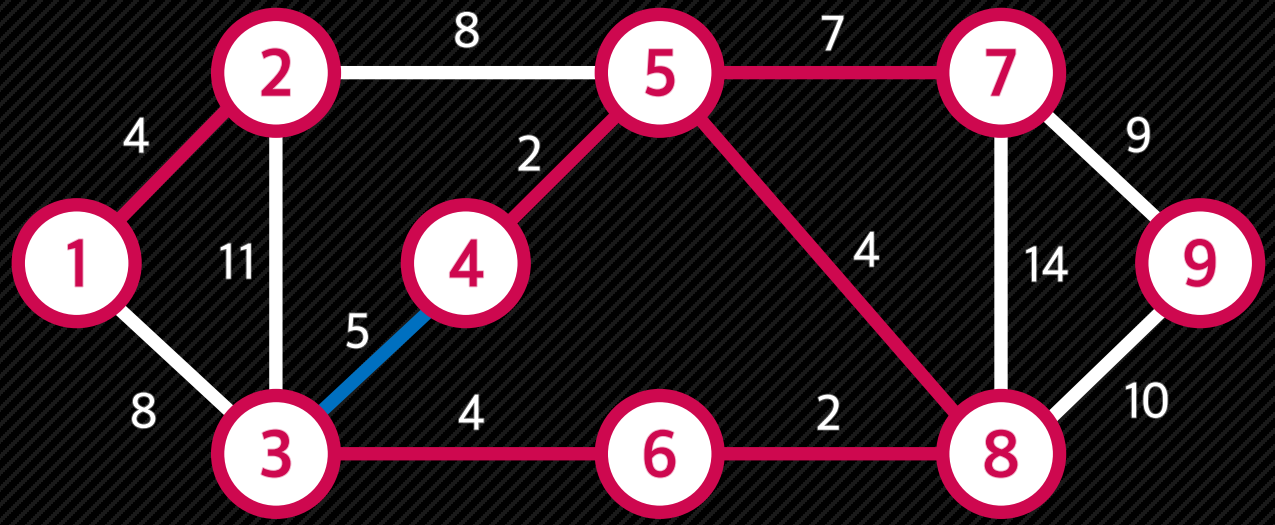



```
function kruskal():
    set mst <- |V| - 1 size of array
    sort edge[] by length
    set union-find initialized to -1
    for each E in edge[]
        if find(u) != find(v)
        push E to mst
        union(u, v)
    return mst
```

	1	2	3	4	5	6	7	8	9
uf	2	-1	8	8	8	8	-1	-1	-1

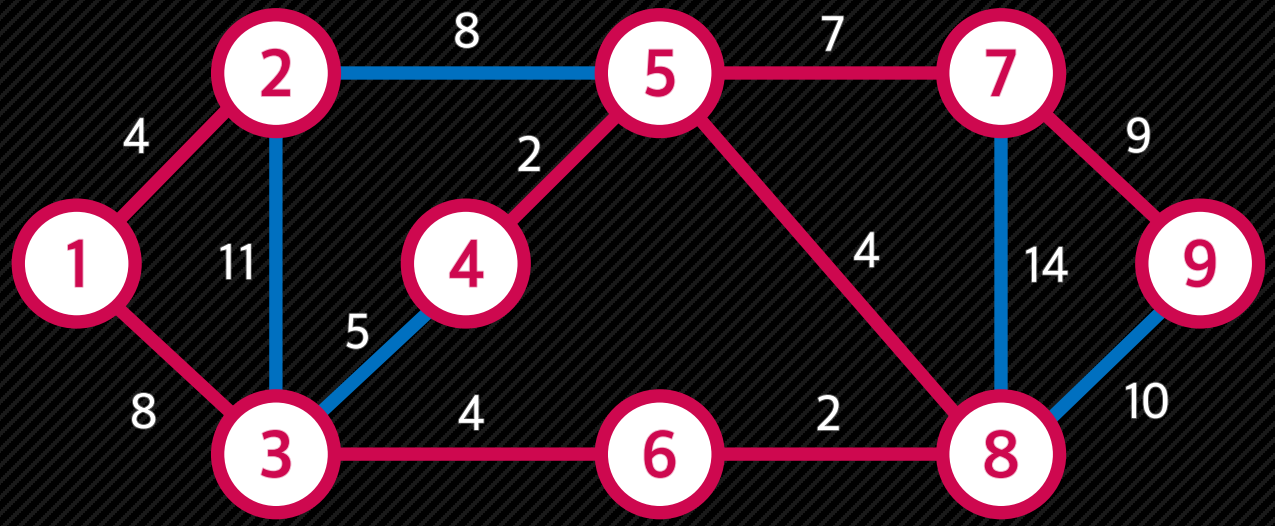
find() 연산을 통한 갱신

Kruskal




```
function kruskal():
    set mst <- |V| - 1 size of array
    sort edge[] by length
    set union-find initialized to -1
    for each E in edge[]
        if find(u) != find(v)
        push E to mst
        union(u, v)
    return mst
```

Kruskal




```
function kruskal():
    set mst <- |V| - 1 size of array
    sort edge[] by length
    set union-find initialized to -1
    for each E in edge[]
        if find(u) != find(v)
        push E to mst
        union(u, v)
    return mst
```

코드로 옮겨 봅시다!

Gold 4 - 최소 스패닝 트리 (#1197)

요약

- 그래프가 주어졌을 때, 그 그래프의 최소 스패닝 트리를 구하는 프로그램을 작성하시오.

제약조건

- 정점의 수의 범위는 1 <= |V| <= 10,000 이다.
- 간선의 수의 범위는 1 <= |E| <= 100,000 이다.

실습

Gold 4 - 우주신과의 교감 (#1774)

?

문제를 직접 읽어 볼까요?

왜? 스패닝 트리 문제일까요?

추가 추천 문제

Gold 3 - 다리 만들기

- 구현에 포커스를 맞춘 문제이고, 오늘 배운 건 양념장 같은 느낌이네요.

Gold 2 - 친구 네트워크

- 일반적인 Union-Find 자료구조로 문제를 풀기엔 특수한 조건이 하나 추가되어 있습니다.

Level 3 - 섬 연결하기

- 대체 왜 이 문제의 카테고리가 Greedy인지는 저도 모르겠습니다.

Level 4 - 호텔 방 배정

- 처음 접하는 입장에선 엄청나게 어려운 문제이지만, 못 풀어도 '왜 Union-Find'인지를 생각해보면 좋을 것 같아요.

“Any Question?”