THE THE THE KRUSKAL

2021.05.19

현경 주희

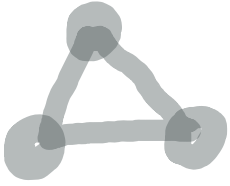
MST

- **신장 트리 (Spanning Tree)**

n개의 정점으로 이루어진 무향 그래프에서 n개의 정점과 n-1개의 간선으로 이루어진 트리

즉, 그래프의 모든 노드가 연결되어 있으면서 트리의 속성을 만족하는 그래프

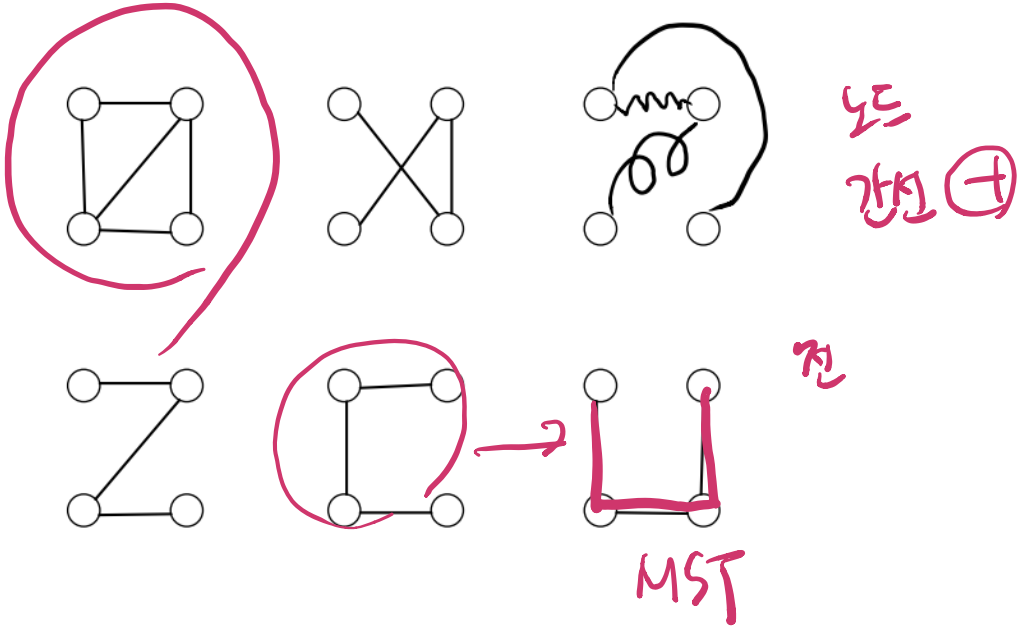
다음중 신장 트리가 아닌 것을 고르시오(?)



- **최소신장트리 (Minimum Spanning Tree)**

무향 가중치 그래프에서 신장 트리를 구성하는 간선들의 가중치의 합이 최소인 신장 트리

MST



MST

최소 신장 트리

왜 필요할까?

- 전기를 여러 개의 집이 있는 동네에 연결시키는 경우
  - 가장 효율적으로 정점들을 모두 연결된 상태로 만들기 위해 필요한 간선의 개수는?
    ~= Tree를 만들기 위한 간선의 개수
  - 그렇다면 최소 비용으로 연결하려면 어떻게 할까??? => MST

- 그래프에서 최소 비용 문제

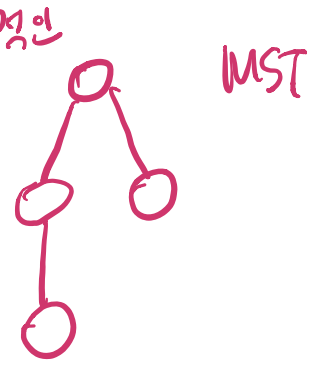
1. 모든 정점을 연결하는 간선들의 가중치의 합이 최소가 되는 트리

2. 두 정점 사이의 최소 비용의 경로 찾기

Kruskal's Algorithm

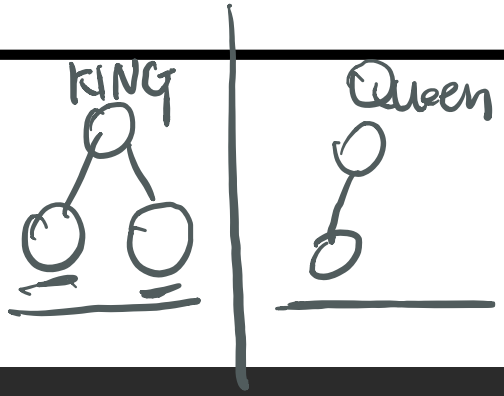
- **간선을 하나씩 선택해서 MST를 찾는 알고리즘**

0. 모든 정점을 독립적인 집합으로 만든다(생각한다)
1. 최초, 모든 간선을 가중치(비용)에 따라 오름차순으로 정렬
2. 가중치가 가장 낮은 간선부터 선택하면서 트리를 증가시킴
   - 사이클이 존재하면 다음으로 가중치가 낮은 간선 선택
   - 그렇다면 사이클이 존재하는지를 어떻게 확인할 것인가?
     - 이를 위해 대표 원소를 찾는 **상호배타집합** 개념을 끌어온다
3. n-1개의 간선이 선택될 때까지 2를 반복



상호배타집합

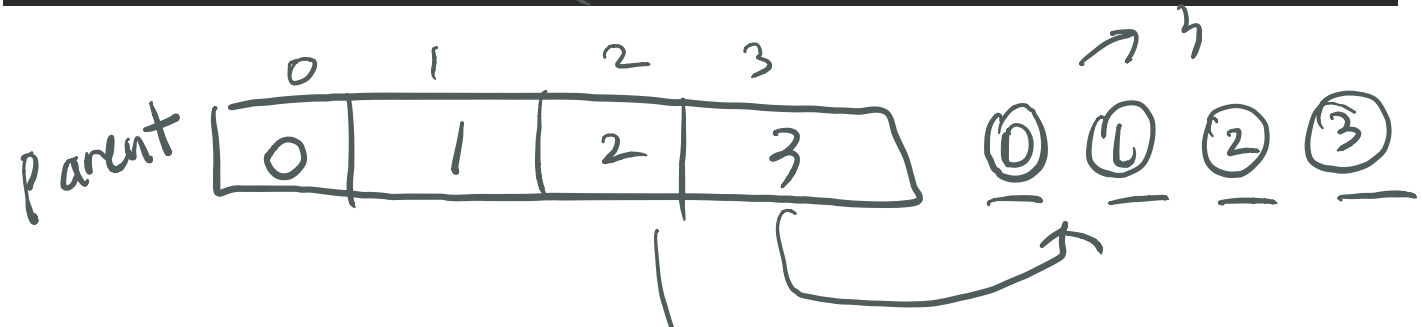
- 하나의 집합(a disjoint set)을 하나의 트리로 표현
- 자식 노드가 부모 노드를 가리키며, 루트 노드가 대장



Make-Set(x) : 초기화

```
def make_set(V):
    for i in range(V+1):
        p[i] = i
```

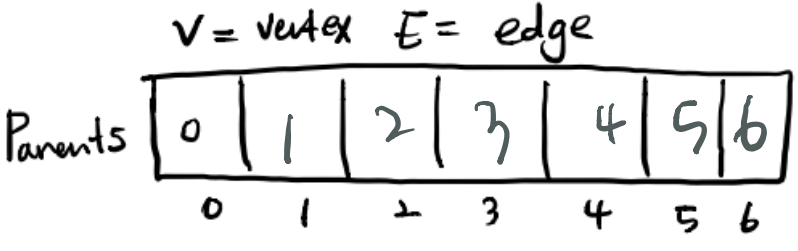
Make set
Find set
Union set



상호배타집합

- 하나의 집합(a disjoint set)을 하나의 트리로 표현
- 자식 노드가 부모 노드를 가리키며, 루트 노드가 대장

Make-Set(x) : 초기화



상호배타집합

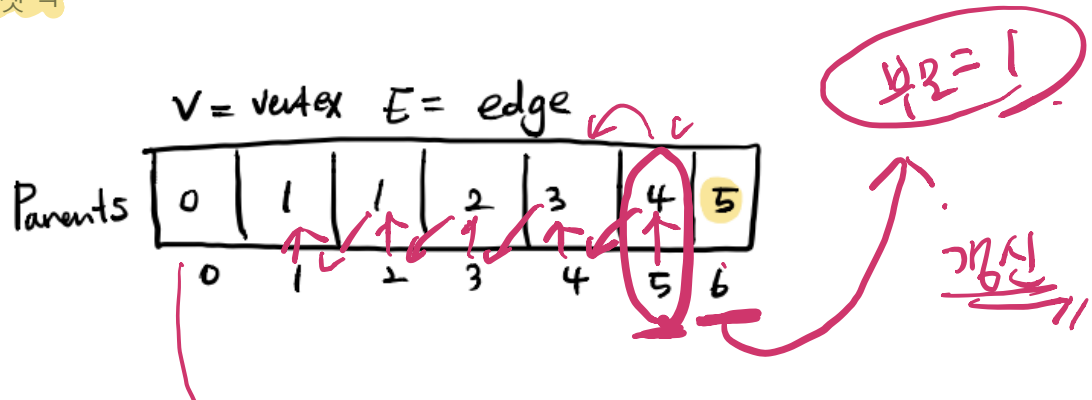
Find_Set(x): x를 포함하는 집합을 찾는 연산

부모를찾는작업

```
def find_set(x):
    if p[x] != x:
        p[x] = find_set(p[x])
    return p[x]  # 거슬러 올라가보기
```

재귀

앞으로의 작업을 효율적으로 만들어 줄 수 있는
효율이 고려된 파인드 셋 ㅋ

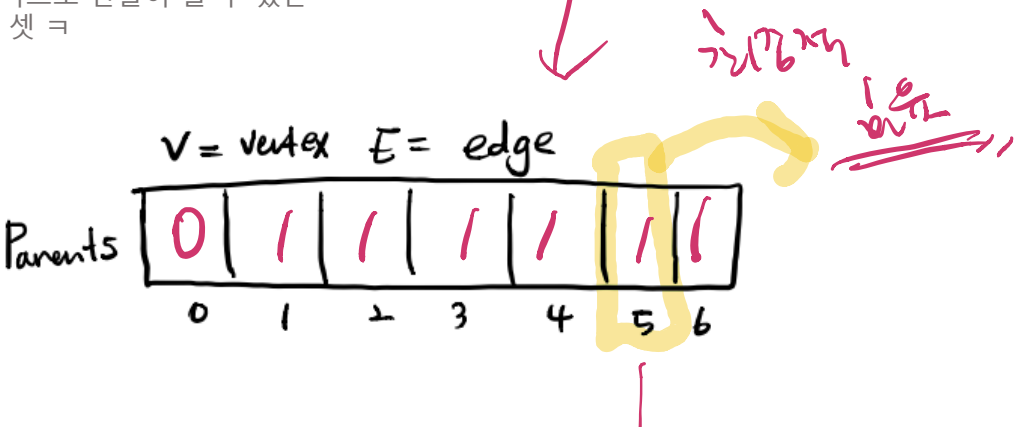


상호배타집합

Find_Set(x): x를 포함하는 집합을 찾는 연산

```
def find_set(x):
    if p[x] != x:
        p[x] = find_set(p[x])
    return p[x]  # 거슬러 올라가보기
```

앞으로의 작업을 효율적으로 만들어 줄 수 있는
효율이 고려된 파인드 셋 ㅋ

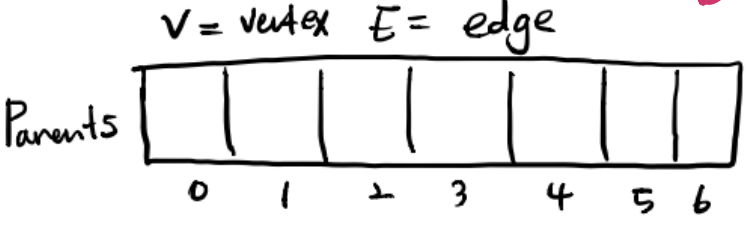


상호배타집합

Find_Set(x): x를 포함하는 집합을 찾는 연산

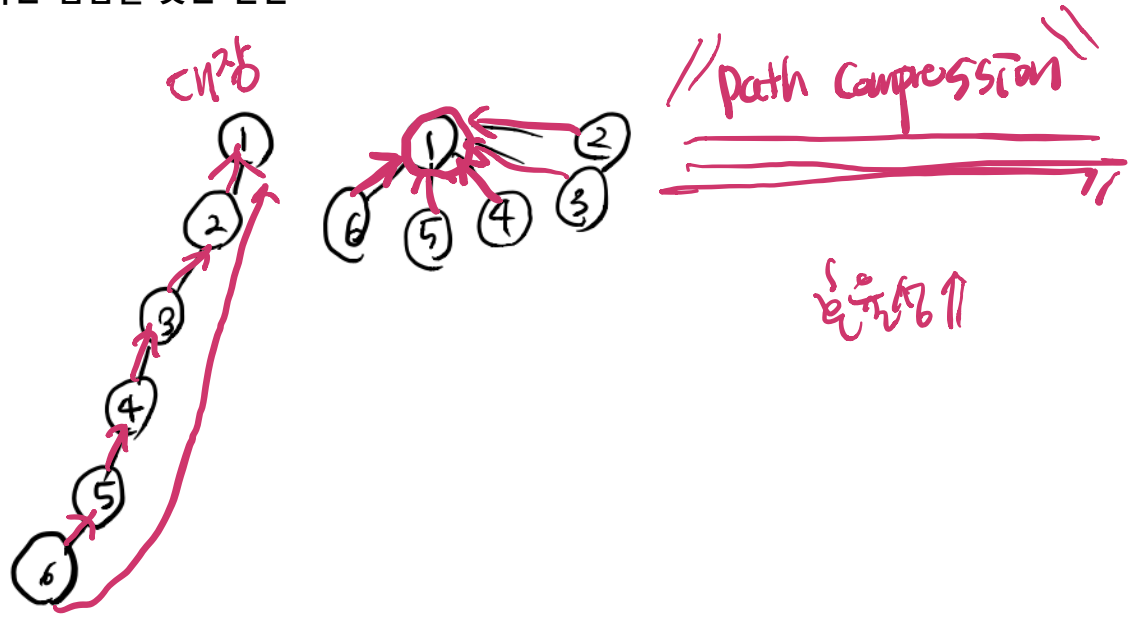
```
def find_set(x):
    if p[x] != x:
        p[x] = find_set(p[x])
    return p[x]  # 거슬러 올라가보기
```

앞으로의 작업을 효율적으로 만들어 줄 수 있는
효율이 고려된 파인드 셋 ㅋ



상호배타집합

Find_Set(x): x를 포함하는 집합을 찾는 연산

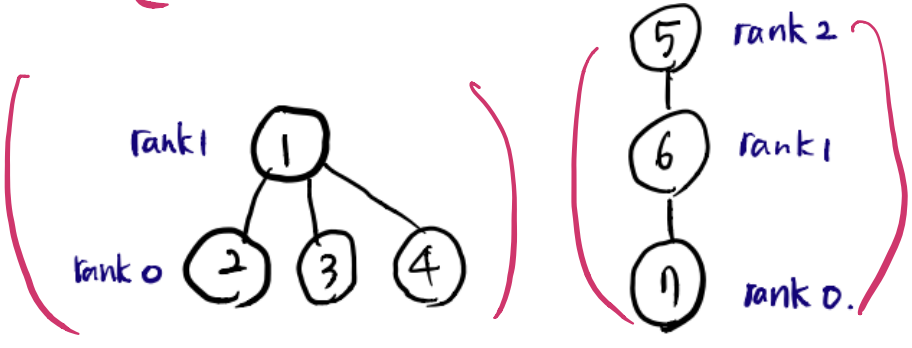


상호배타집합

Union_Set(x): x와 y를 포함하는 두 집합을 통합하는 연산 (랭크 고려 X)

```
def union(x, y):
    p[find_set(y)] = find_set(x)
```

뮤조건 합해버린다

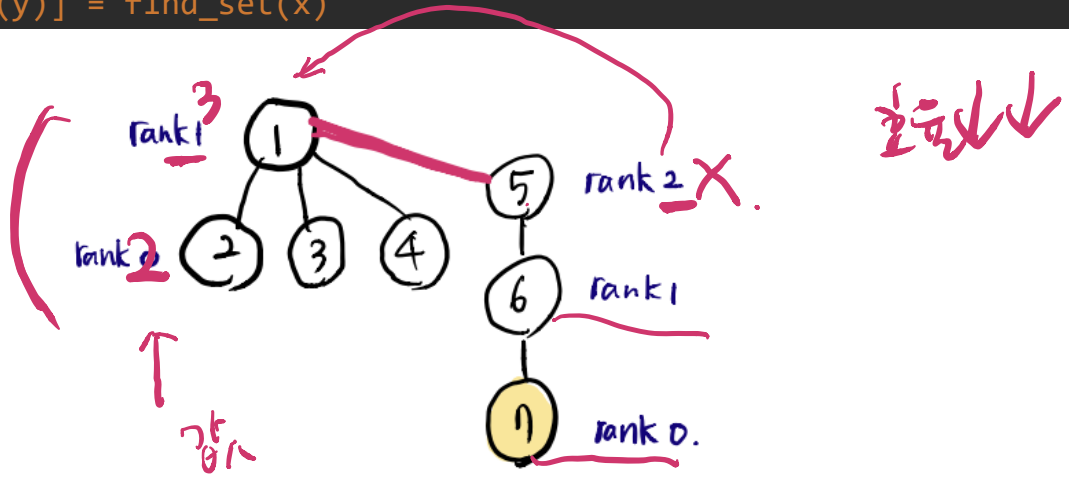


상호배타집합

Union_Set(x): x와 y를 포함하는 두 집합을 통합하는 연산 (랭크 고려 X)

```
def union(x, y):
    p[find_set(y)] = find_set(x)
```

뮤조건 합해버린다



상호배타집합

Union_Set(x): x와 y를 포함하는 두 집합을 통합하는 연산 (랭크 고려 O)

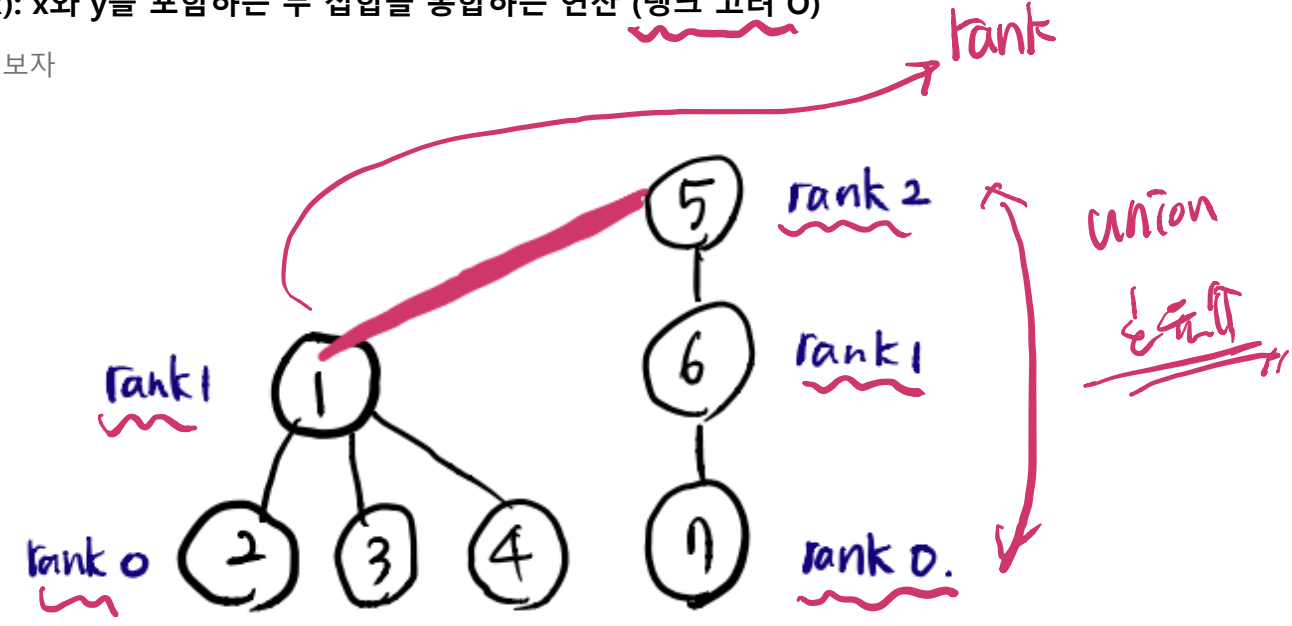
```
def union(x, y):
    link(find_set(x), find_set(y))

def link(x, y):
    if r[x] > r[y]:
        p[y] = x
    else:
        p[x] = y
        if r[x] == r[y]:
            r[y] += 1
```

랭크를 고려해보자

상호배타집합

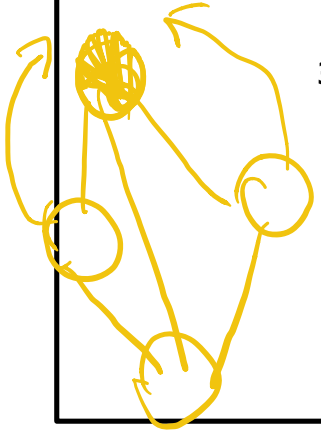
Union_Set(x): x와 y를 포함하는 두 집합을 통합하는 연산 (랭크 고려 O)

랭크를 고려해보자



Kruskal's Algorithm

1. 모든 정점을 독립적인 집합으로 만든다
2. 모든 간선을 비용을 기준으로 오름차순으로 정렬하고,
   비용이 작은 간선부터 양 끝의 두 정점을 비교
3. 두 정점의 최상위 정점을 확인, 서로 다를 경우 두 정점을 연결
   (사이클 없다)

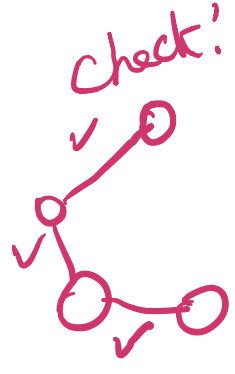


탐욕 알고리즘을 기초로 하고 있음

(당장 눈 앞의 최소 비용을 선택, 결과적으로 최적의 솔루션)

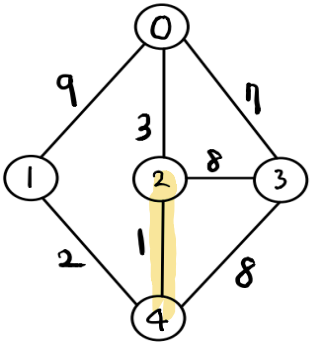
Kruskal's Algorithm

```
MST-KRUSKAL(G, W)

    A <- 0                          // 공집합
    For vertex in G.V               //  G.V: vertices
            Make_Set(v)             //  G.E: edges

    G.E에 포함된 간선들을 가중치 W에 의해 정렬

    For 가중치가 가장 낮은 간선 (u, v) ∈ G.E 선택(n-1)개
        IF Find_Set(u) ≠ Find_Set(v)
        A <- A ∪ {(u, v)}
        Union(u, v);

    RETURN A
```



Kruskal's Algorithm

해보자!



	w	
2-4	1	✓
1-4	2	
0-2	3	
0-3	7	
2-3	8	
3-4	8	
0-1	9	

SWEA 5249 tc 2

2-4.

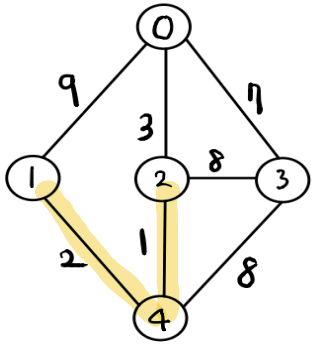
	0	1	2	3	4
parent	0	1	4	3	4
rank	0	0	0	0	1

++

ans 1

Kruskal's Algorithm

해보자!



	w
2-4	1
1-4	2
0-2	3
0-3	7
2-3	8
3-4	8
0-1	9

→ 2

1 — 4

Findset

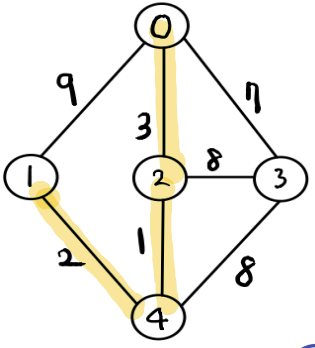
	0	1	2	3	4
parent	0	4	4	3	4
rank	0	0	0	0	1

→ 크기 11

ans 1 + 2

Kruskal's Algorithm

해보자!



	w
2-4	1
1-4	2
0-2	3
0-3	7
2-3	8
3-4	8
0-1	9

3

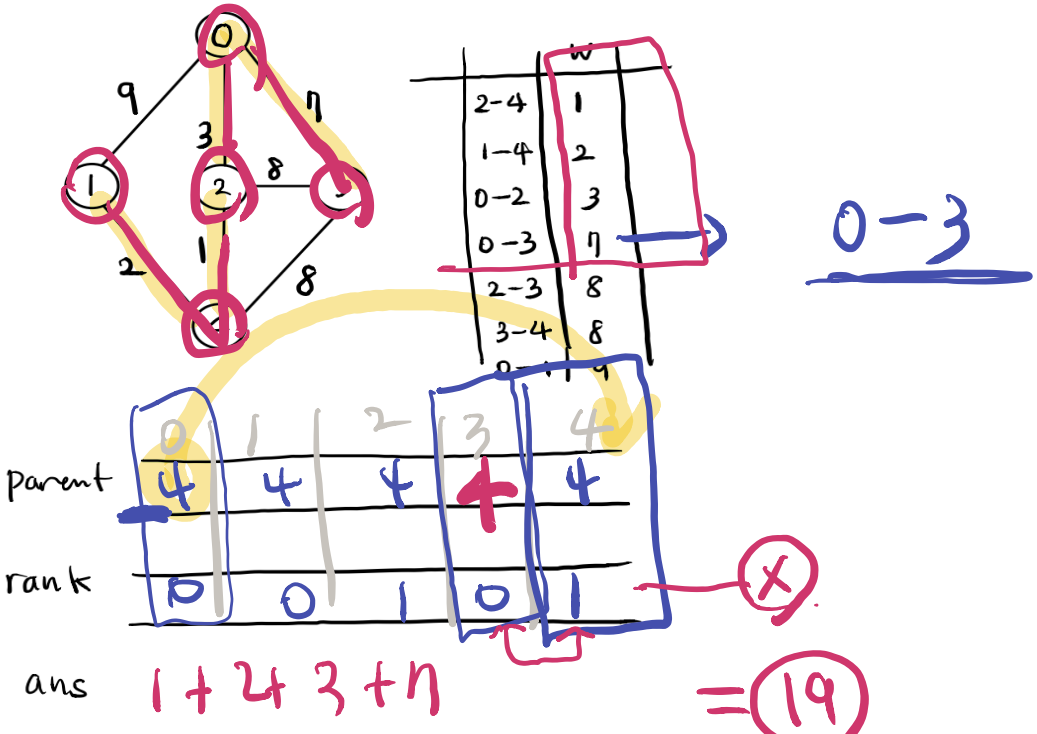
0-2

	0	1	2	3	4
parent	4	4	4	3	4
rank	0	0	1	0	1

ans 1+2+3

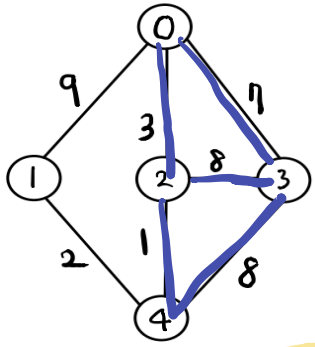
Kruskal's Algorithm

해보자!



Kruskal's Algorithm

해보자!



	w
2-4	1
1-4	2
0-2	3
0-3	7
2-3	8
3-4	8
0-1	9

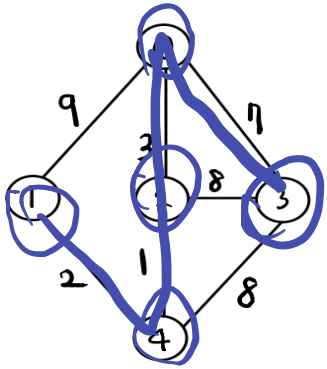
check

	0	1	2	3	4
parent	4	4	4	4	4
rank	0	0	1	0	1

ans

Kruskal's Algorithm

해보자!



	w
2-4	1
1-4	2
0-2	3
0-3	7
2-3	8
3-4	8
0-1	9