MST

MST(Minimum Spanning Tree)란

- Spanning Tree중에서 사용된 간선들의 가중치 합이 최소인 트리
- 간선의 가중치를 고려하여 최소 비용의 Spanning Tree를 선택하는 것을 말한다.
- 그래프에 있는 모든 정점들을 가장 적은 수의 간선과 비용으로 연결하는 것

MST의 특징

- 간선의 가중치의 합이 최소여야 한다.
- n개의 정점을 가지는 그래프에 대해 반드시 (n-1)개의 간선만을 사용해야 한다.
- 사이클이 포함되어서는 안 된다.

MST의 구현방법

1. **Kruskal MST 알고리즘**

   Greedy Method를 이용하여 그래프의 모든 정점을 최소 비용으로 연결하는 최적 해답을 구하는 것

   [과정]

   1. 그래프의 간선들을 가중치의 오름차순으로 정렬한다.
   2. 정렬된 간선 리스트에서 순서대로 사이클을 형성하지 않는 간선들을 선택한다.
      A. 즉, 가장 낮은 가중치를 선택한다.
      B. 사이클을 형성하는 간선을 제외한다.
   3. 해당 간선을 현재의 MST의 집합에 추가한다.

2. **Prim MST 알고리즘**

시작 정점에서부터 출발하여 Spanning Tree 집합을 단계적으로 확장해 나가는 방법

[과정]

1. 시작 단계에서는 시작 정점만이 MST 집합에 포함된다.
2. 앞 단계에서 만들어진 MST집합에 인접한 정점들 중에서 최소 간선으로 연결된 정점을 선택하여 트리를 확장한다.
   A. 즉, 가장 낮은 가중치를 먼저 선택한다.
3. 위의 과정을 트리가 (N-1)개의 간선을 가질 때까지 반복한다.

Kruskal

Kruskal 알고리즘이란

Greedy Method를 이용하여 그래프의 모든 정점을 최소 비용으로 연결하는 최적 해답을 구하는 것

- Greedy Method
  - 결정을 해야 할 때마다 그 순간에 가장 좋다고 생각되는 것을 선택함으로써 최종적인 해답에 도달하는 것
  - 그 순간에는 최적이지만, 전체적인 관점에서 최적이라는 보장이 없기 때문에 반드시 검증해야 한다.
  - 다행히 Kruskal 알고리즘은 최적의 해답을 주는 것으로 증명되어 있다.
- MST(최소 비용 신장 트리)가 *1) 최소 비용의 간선으로 구성됨, 2) 사이클을 포함하지 않음* 의 조건에 근거하여 **각 단계에서 사이클을 이루지 않는 최소 비용 간선을 선택**한다.

Kruskal 알고리즘의 동작

1. 그래프의 간선들을 가중치의 오름차순으로 정렬한다.

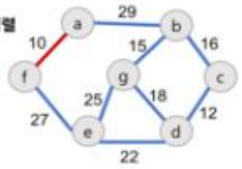
2. 정렬된 간선 리스트에서 순서대로 사이클을 형성하지 않는 간선을 선택한다.
    A. 즉, 가장 낮은 가중치를 먼저 선택한다.
    B. 사이클을 형성하는 간선을 제외한다.
3. 해당 간선을 현재의 MST의 집합에 추가한다.

Kruskal 알고리즘의 구체적인 동작 과정

Kruskal 알고리즘을 이용하여 MST를 만드는 과정

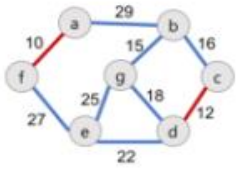
- **간선 선택을 기반으로** 하는 알고리즘
- 이전 단계에서 만들어진 신장 트리와는 상관없이 무조건 최소 간선만을 선택하는 방법

1) 간선들의 가중치 오름차순 정렬



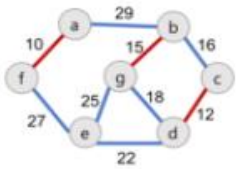
간선의 가중치	af	cd	bg	bc	dg	de	eg	ef	ab
	10	12	15	16	18	22	25	27	29

2)



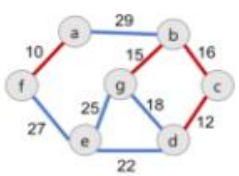
간선의 가중치	af	cd	bg	bc	dg	de	eg	ef	ab
	10	12	15	16	18	22	25	27	29

3)



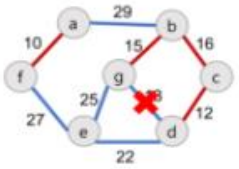
간선의 가중치	af	cd	bg	bc	dg	de	eg	ef	ab
	10	12	15	16	18	22	25	27	29

4)



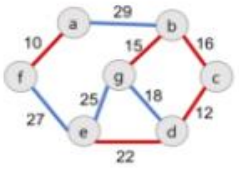
간선의 가중치	af	cd	bg	bc	dg	de	eg	ef	ab
	10	12	15	16	18	22	25	27	29

5) 사이클 형성. dg는 제외



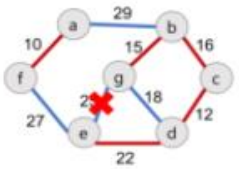
간선의 가중치	af	cd	bg	bc	dg	de	eg	ef	ab
	10	12	15	16	18	22	25	27	29

6)



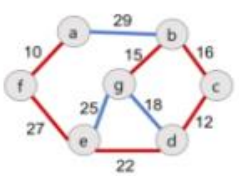
간선의 가중치	af	cd	bg	bc	dg	de	eg	ef	ab
	10	12	15	16	18	22	25	27	29

7) 사이클 형성. eg는 제외



간선의 가중치	af	cd	bg	bc	dg	de	eg	ef	ab
	10	12	15	16	18	22	25	27	29

8) N-1개의 간선 생성. 종료



간선의 가중치	af	cd	bg	bc	dg	de	eg	ef	ab
	10	12	15	16	18	22	25	27	29

주의

1. 다음 간선을 이미 선택된 간선들의 집합에 추가할 때 **사이클을 생성하는지를 체크**
    A. 새로운 간선이 이미 다른 경로에 의해 연결되어 있는 정점들을 연결할 대 사이클이 형성된다.
    B. 즉, 추가할 새로운 간선의 양 끝 정점이 같은 집합에 속해 있으면 사이클이 형성된다.
2. **사이클 생성 여부를 확인하는 방법**
    A. 추가하고자 하는 간선의 양끝 정점이 같은 집합에 속해 있는지를 먼저 검사해야 한다.
    B. **union-find 알고리즘** 이용

Union-find 알고리즘

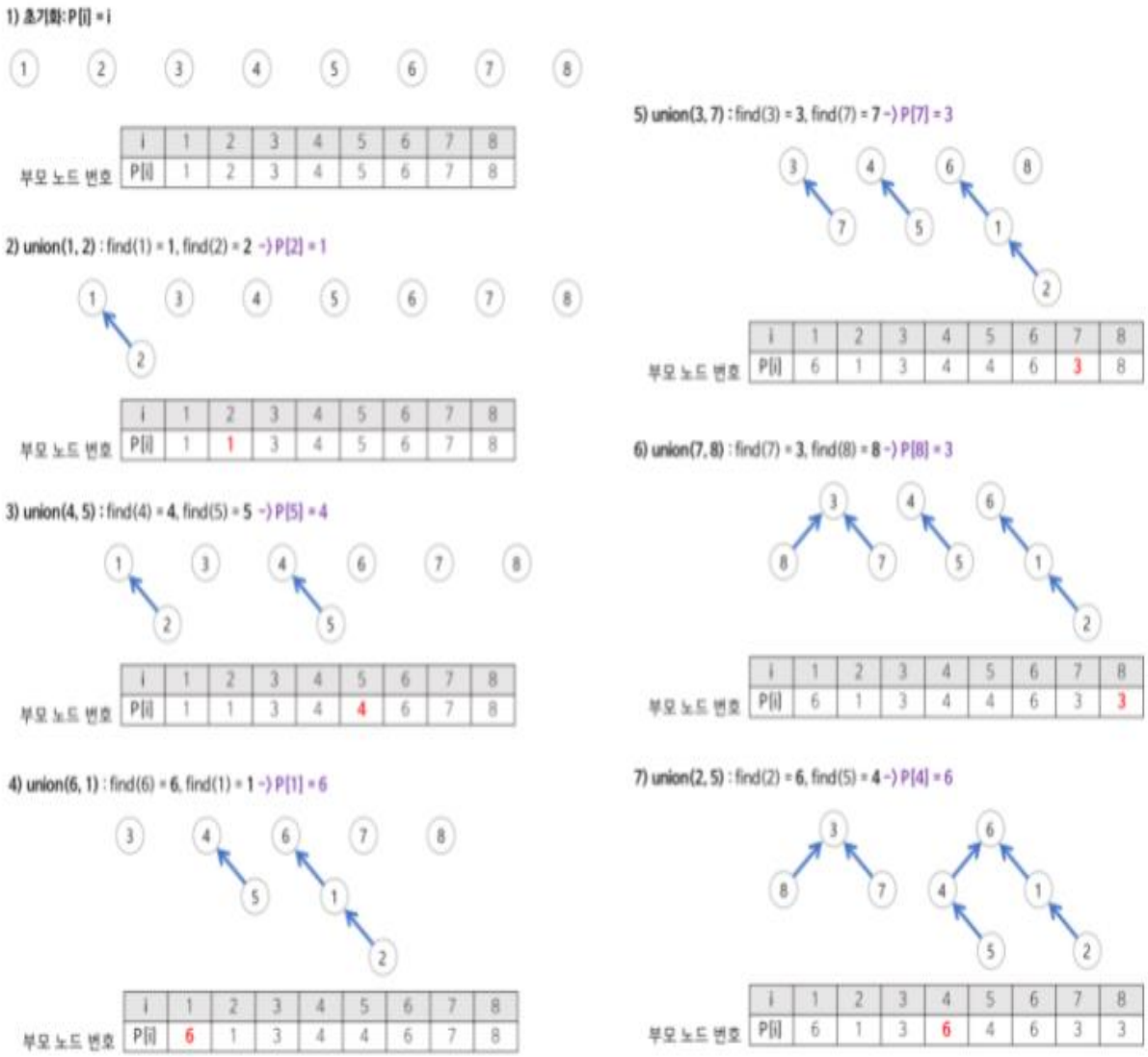
여러 개의 노드가 존재할 때 두 개의 노드를 선택해서, 현재 이 두 노드가 서로 같은 그래프에 속하는지 판별하는 알고리즘

Disjoint Set을 표현할 때 사용하는 알고리즘

- Disjoint Set이란
    - 서로 중복되지 않는 부분 집합들로 나눠진 원소들에 대한 정보를 저장하고 조작하는 자료구조
    - 즉, 공통 원소가 없는, '상호 배타적'인 부분 집합들로 나눠진 원소들에 대한 자료구조이다
- Union-find의 연산
    - union(x, y)
        - 합하기
        - x가 속한 집합과 y가 속한 집합을 합친다. 즉, x와 y가 속한 두 집합을 합치는 연산

- find(x)
  - 찾기
  - x가 속한 집합의 대표값(루트 노드 값)을 반환한다. 즉, x가 어떤 집합에 속해 있는지 찾는 연산

Union-Find의 과정



Union-Find 알고리즘의 코드

미리 해야 할 과정

- 각 노드의 부모 노드를 가리키는 배열을 선언하고, 초기화 한다. (Parent배열)

Find: 루트에 도달할 때까지 재귀를 사용하여 부모 노드를 찾는다.

```
int find(int x) {
        if (parent[x] == x) return x;
        return parent[x] = find(parent[x]);
}
```

Union: x와 y의 부모 노드가 같다면 같은 집합이므로 종료, 아니면 y의 부모를 x로 바꿔준다.

```
void merge(int x, int y) {
        x = find(x);
        y = find(y);
        if (x == y) return;
        parent[y] = x;
}
```

Kruskal 알고리즘의 시간 복잡도

- union-find 알고리즘을 이용하면 Kruskal 알고리즘의 시간 복잡도는 간선들을 정렬하는 시간에 좌우된다.
- 즉, 간선 e개를 퀵 정렬과 같은 효율적인 알고리즘으로 정렬한다면
  - Kruskal 알고리즘의 시간 복잡도는 O($elog_2e$)이 된다.
- Prim 알고리즘의 시간 복잡도는 O(n^2)이므로
  - 그래프 내에 적은 숫자의 간선만을 가지는 '희소 그래프'의 경우 Kruskal 알고리즘이 적합하고
  - 그래프에 간선이 많이 존재하는 '밀집 그래프'의 경우는 Prim알고리즘이 적합하다.

Prim

Prim 알고리즘이란

시작 정점에서부터 출발하여 Spanning Tree 집합을 단계적으로 확장해 나가는 방법

Prim 알고리즘의 동작

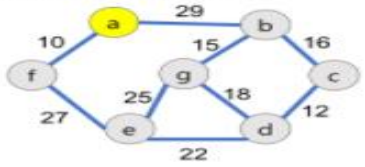
1. 시작 단계에서는 시작 정점만이 MST 집합에 포함된다.
2. 앞 단계에서 만들어진 MST 집합에 인접한 정점들 중에서 최소 간선으로 연결된 정점을 선택하여 트리를 확장한다.
    A. 즉, 가장 낮은 가중치를 먼저 선택한다.
3. 위의 과정을 트리가 (N-1)개의 간선을 가질 때까지 반복한다.

Prim 알고리즘의 구체적인 동작 과정

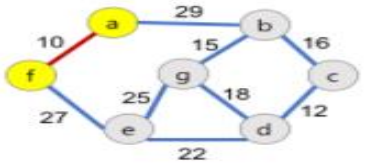
Prim 알고리즘을 이용하여 MST를 만드는 과정

- **정점 선택을 기반**으로 하는 알고리즘
- 이전 단계에서 만들어진 신장 트리를 확장하는 방법

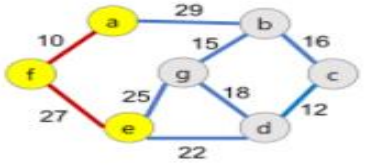
1) 시작 단계: 시작 정점만 포함



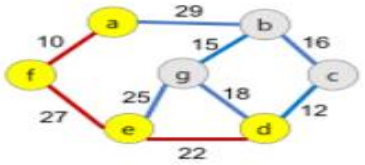
2) 인접 정점 중 최소 간선으로 연결된 정점 선택



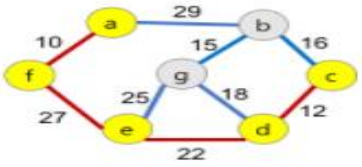
3)



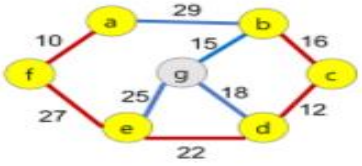
4)



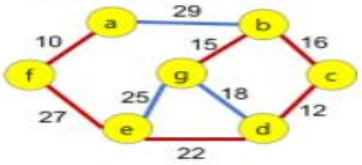
5)



6)



7) N-1개의 간선 생성. 종료



Prim 알고리즘의 시간 복잡도

- 주 반복문이 정점의 수 n만큼 반복하고, 내부 반복문이 n번 반복
- Prim 알고리즘의 시간 복잡도는 O(n^2)이 된다.

MST관련 문제

- 1197 최소 스패닝 트리
- 1922 네트워크 연결