

최소 스패닝 트리

김영균

정의

- 스패닝 트리

무향 그래프의 모든 정점들을 연결하는 부분 그래프 = 트리

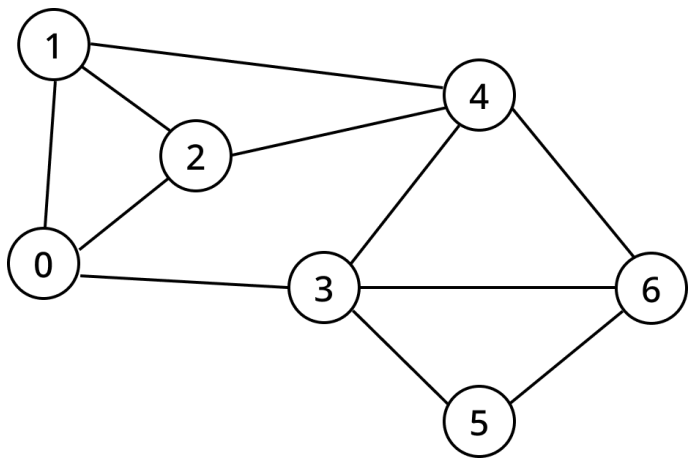


그림1. 무향 그래프

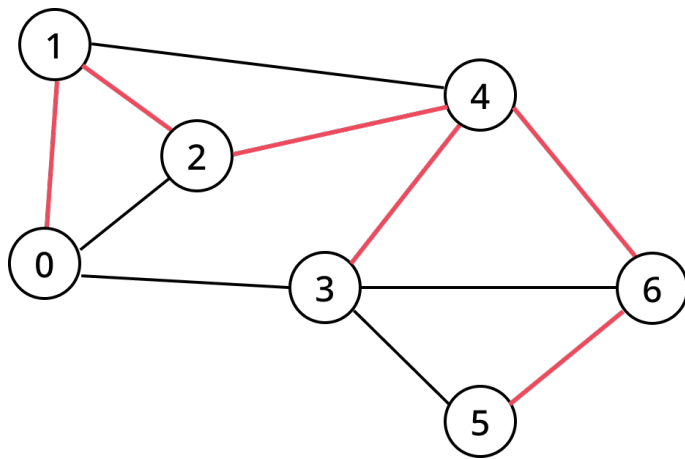


그림2. 올바른 스패닝 트리

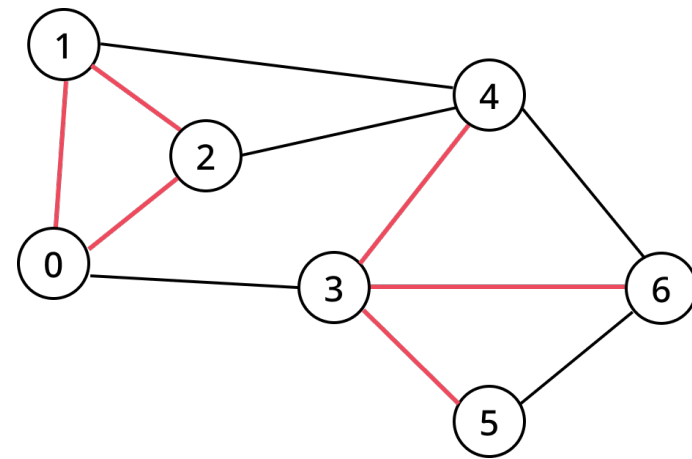


그림3. 잘못된 스패닝 트리

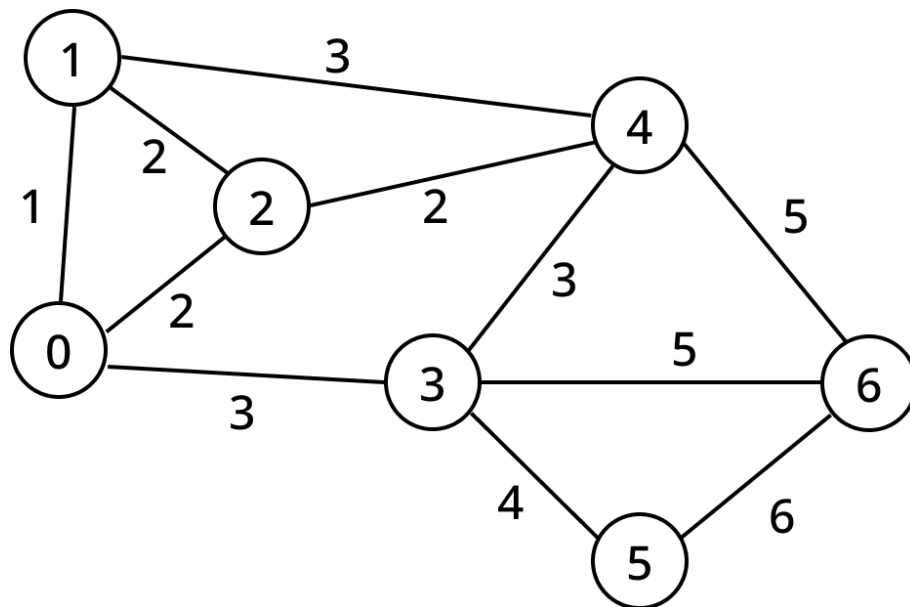
최소 스패닝 트리

- 최소 스패닝 트리

무향 그래프의 스패닝 트리 중에서 가중치의 합이 최소인 트리

- 두 가지 알고리즘

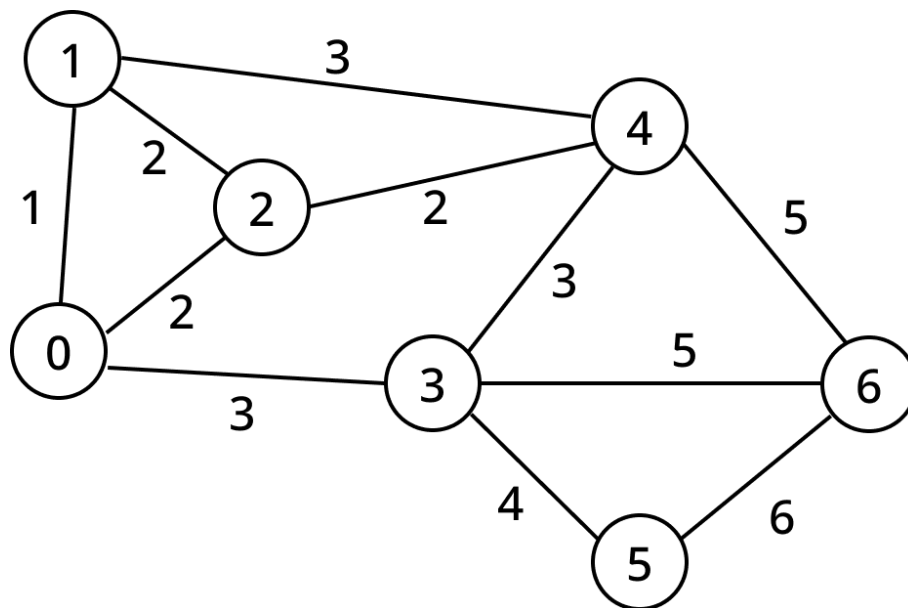
1. 크루스칼 알고리즘
2. 프림 알고리즘



크루스칼 알고리즘

- 최소 스패닝 트리에 포함될 가능성 생각하기

가중치가 큰 간선과 가중치가 작은 간선 중 어떤 간선이 최소 스패닝 트리에 포함될 가능성이 높을까?



크루스칼 알고리즘

- 최소 스패닝 트리에 포함될 가능성 생각하기

가중치가 큰 간선과 가중치가 작은 간선 중 어떤 간선이 최소 스패닝 트리에 포함될 가능성이 높을까?

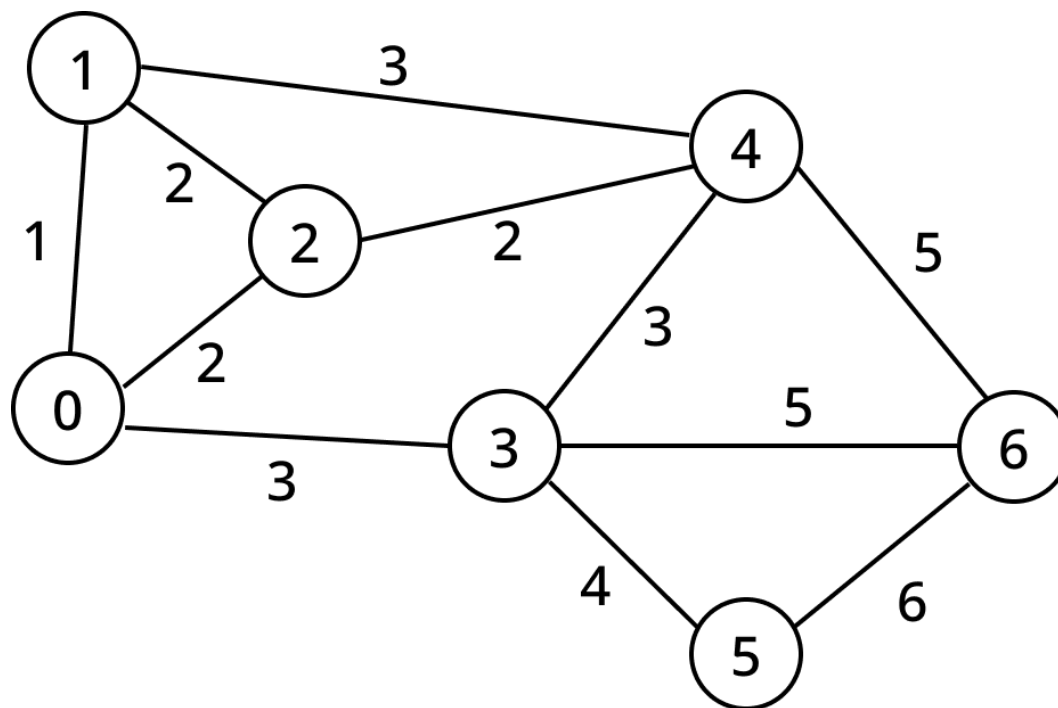
➡ 모든 간선을 가중치 순으로 오름차순 정렬한다.

```
struct Edge {
    int from, to, weight;
    bool operator< (const Edge& e) const {
        return weight < e.weight;
    }
};
vector<Edge> edges;
for(int i = 0; i < E; i++) {
    int from, to, weight;
    cin >> from >> to >> weight;
    edges.push_back({from, to, weight});
}
sort(edges.begin(), edges.end());
```

크루스칼 알고리즘

- 가중치가 작은 간선부터 고를 수 있으면 고른다.

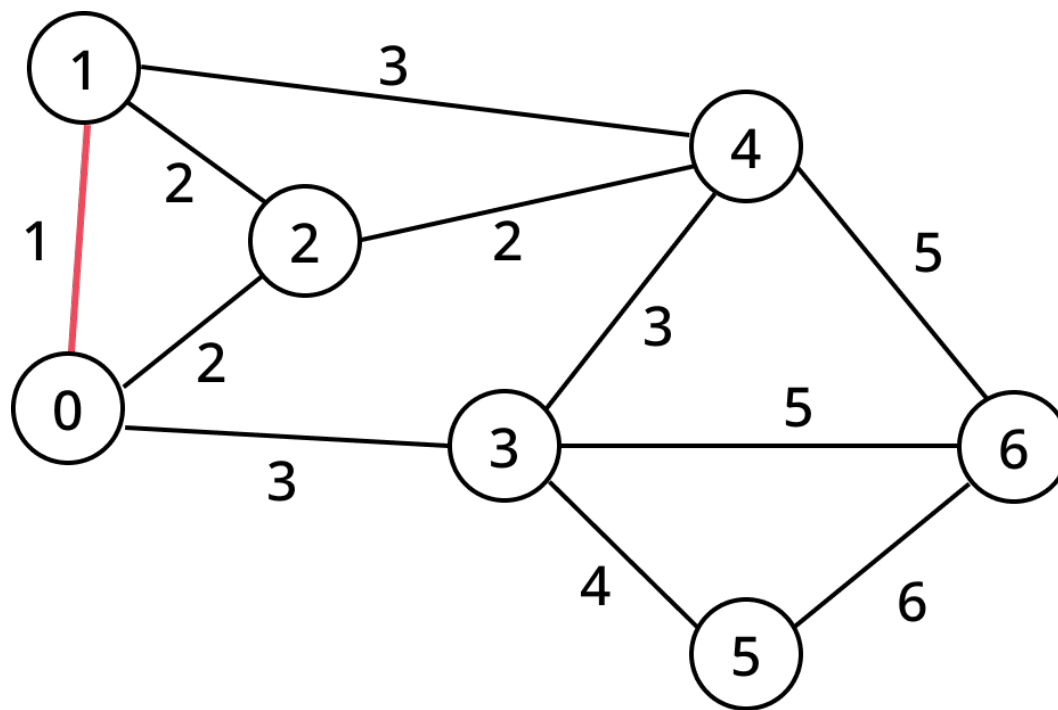
간선을 추가해도 여전히 스패닝 트리를 만족하는 간선을 고른다.



크루스칼 알고리즘

- 가중치가 작은 간선부터 고를 수 있으면 고른다.

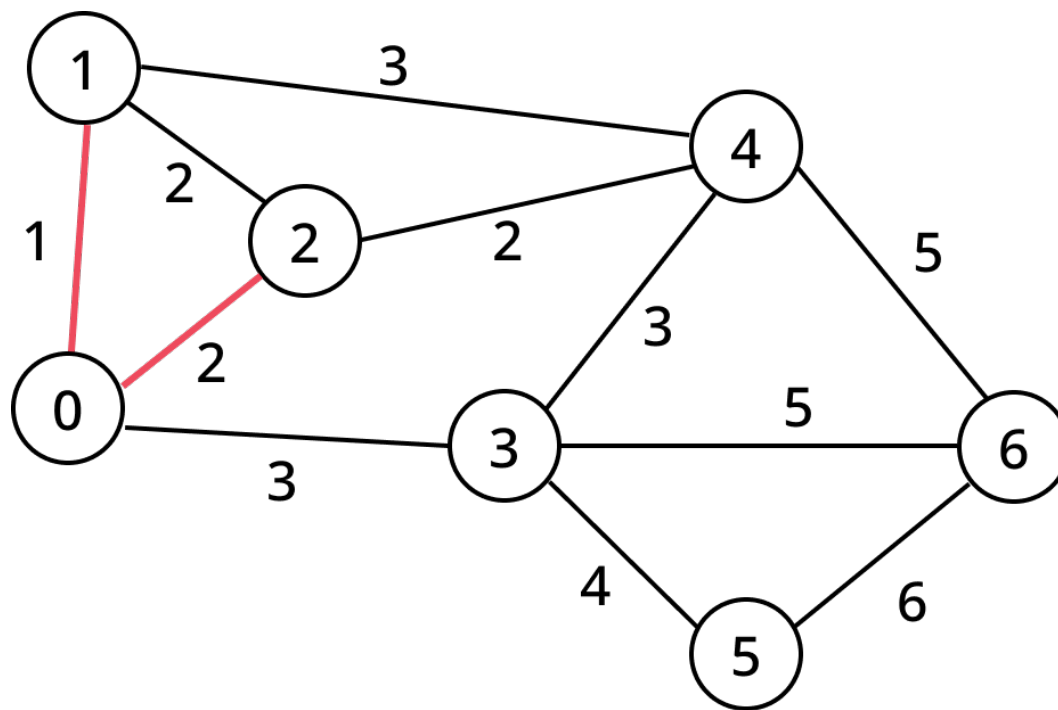
간선을 추가해도 여전히 스패닝 트리를 만족하는 간선을 고른다.



크루스칼 알고리즘

- 가중치가 작은 간선부터 고를 수 있으면 고른다.

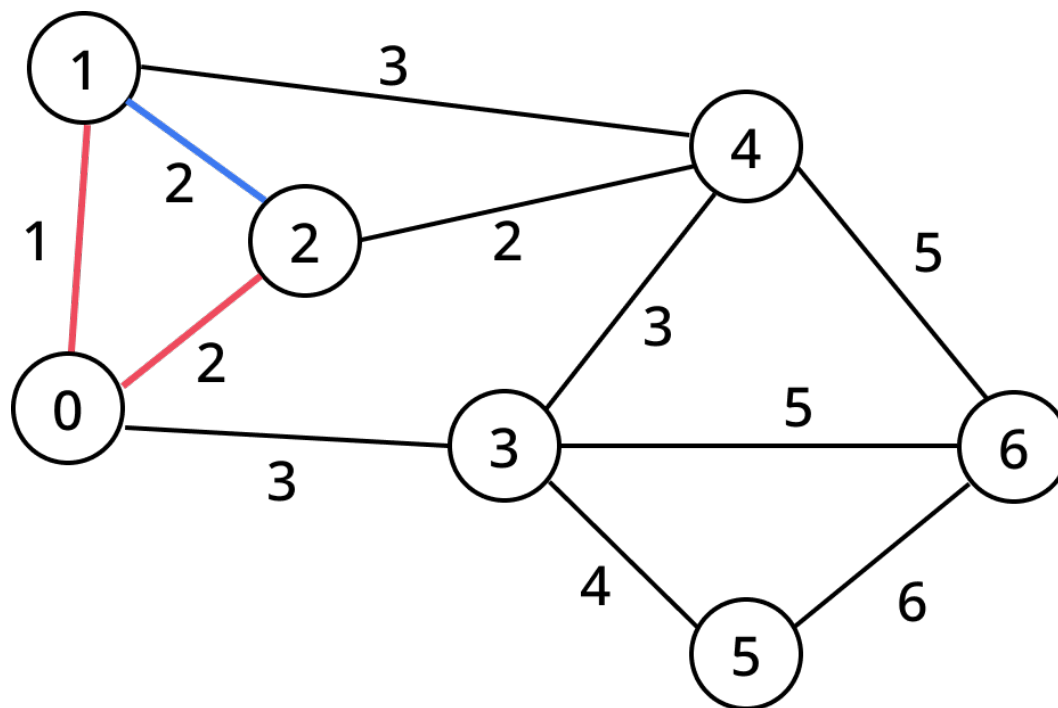
간선을 추가해도 여전히 스패닝 트리를 만족하는 간선을 고른다.



크루스칼 알고리즘

- 가중치가 작은 간선부터 고를 수 있으면 고른다.

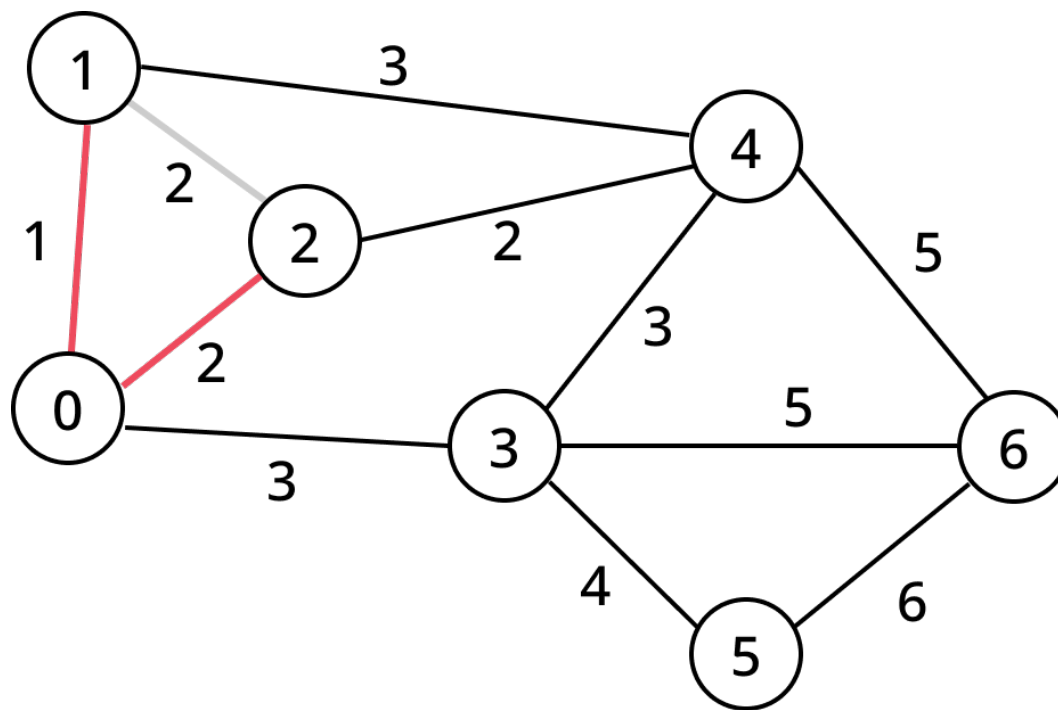
간선을 추가해도 여전히 스패닝 트리를 만족하는 간선을 고른다.



크루스칼 알고리즘

- 가중치가 작은 간선부터 고를 수 있으면 고른다.

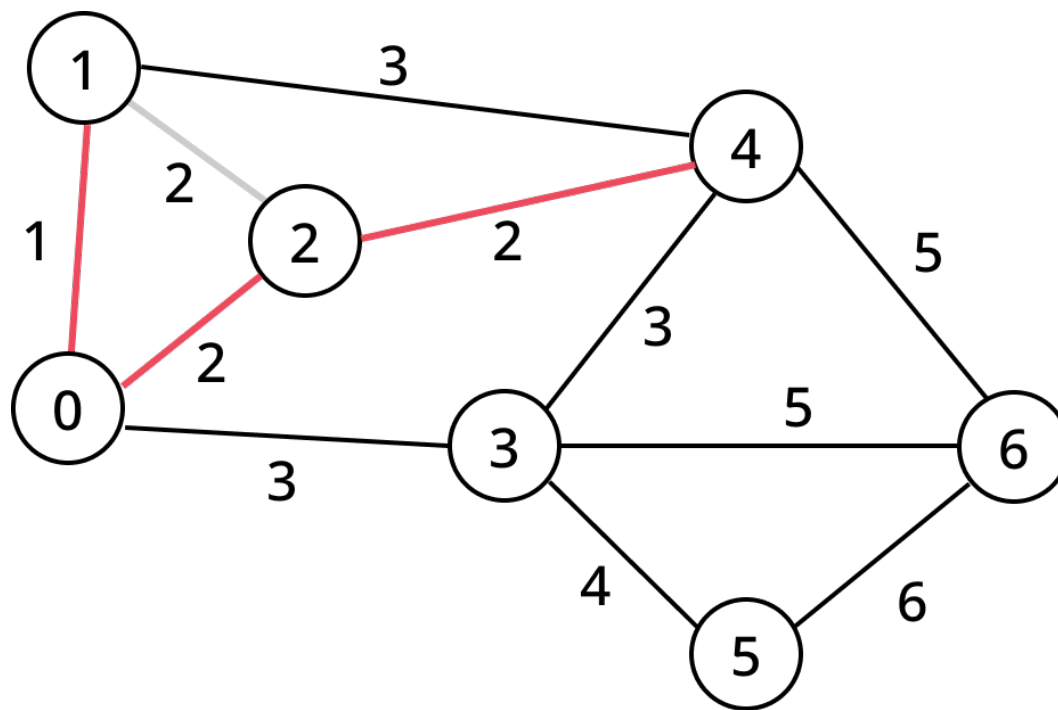
간선을 추가해도 여전히 스패닝 트리를 만족하는 간선을 고른다.



크루스칼 알고리즘

- 가중치가 작은 간선부터 고를 수 있으면 고른다.

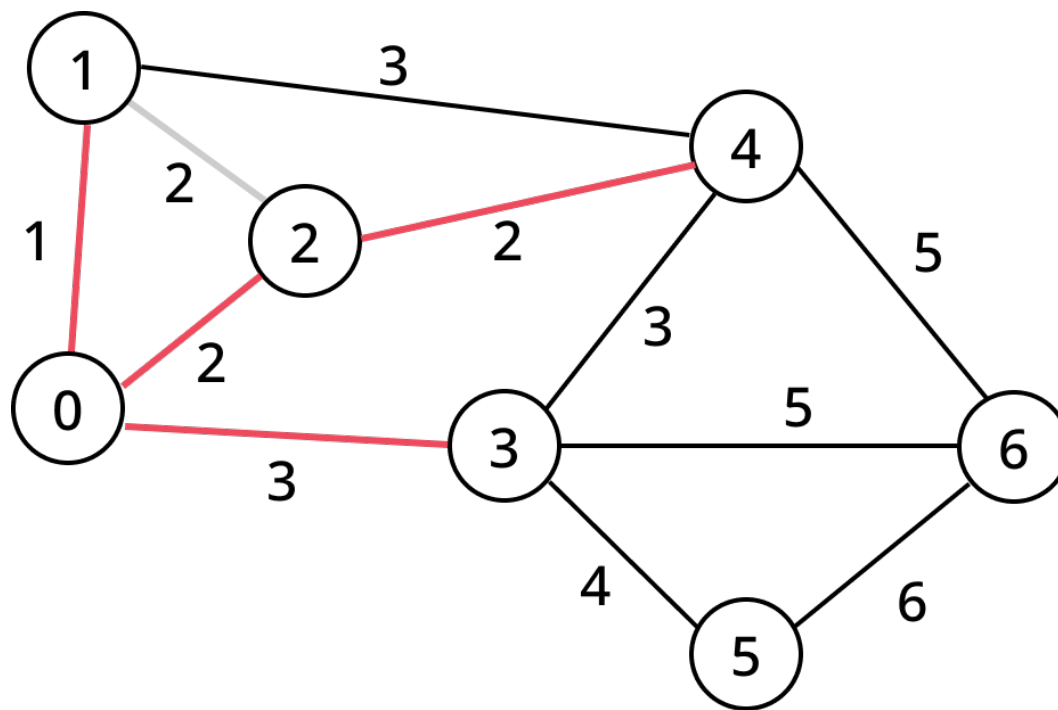
간선을 추가해도 여전히 스패닝 트리를 만족하는 간선을 고른다.



크루스칼 알고리즘

- 가중치가 작은 간선부터 고를 수 있으면 고른다.

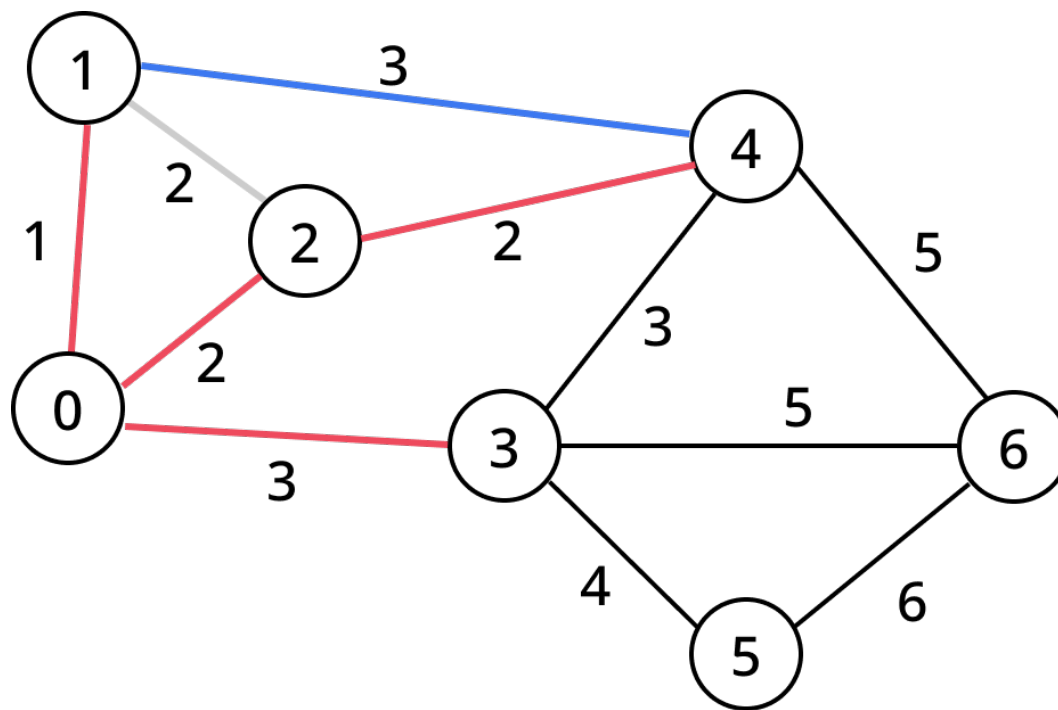
간선을 추가해도 여전히 스패닝 트리를 만족하는 간선을 고른다.



크루스칼 알고리즘

- 가중치가 작은 간선부터 고를 수 있으면 고른다.

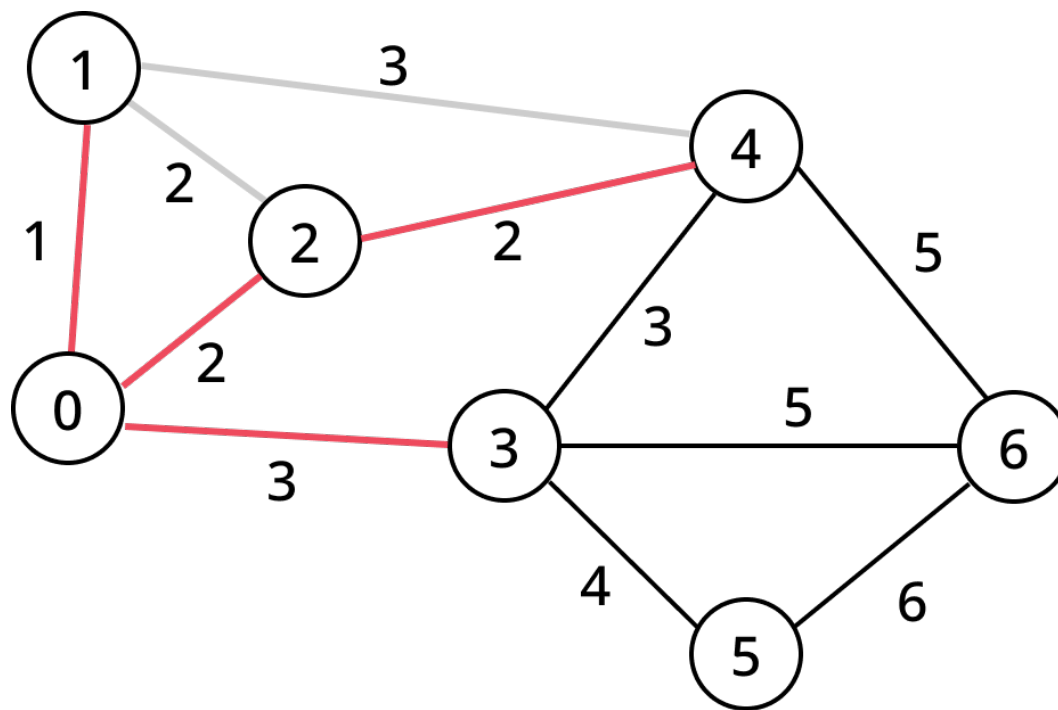
간선을 추가해도 여전히 스패닝 트리를 만족하는 간선을 고른다.



크루스칼 알고리즘

- 가중치가 작은 간선부터 고를 수 있으면 고른다.

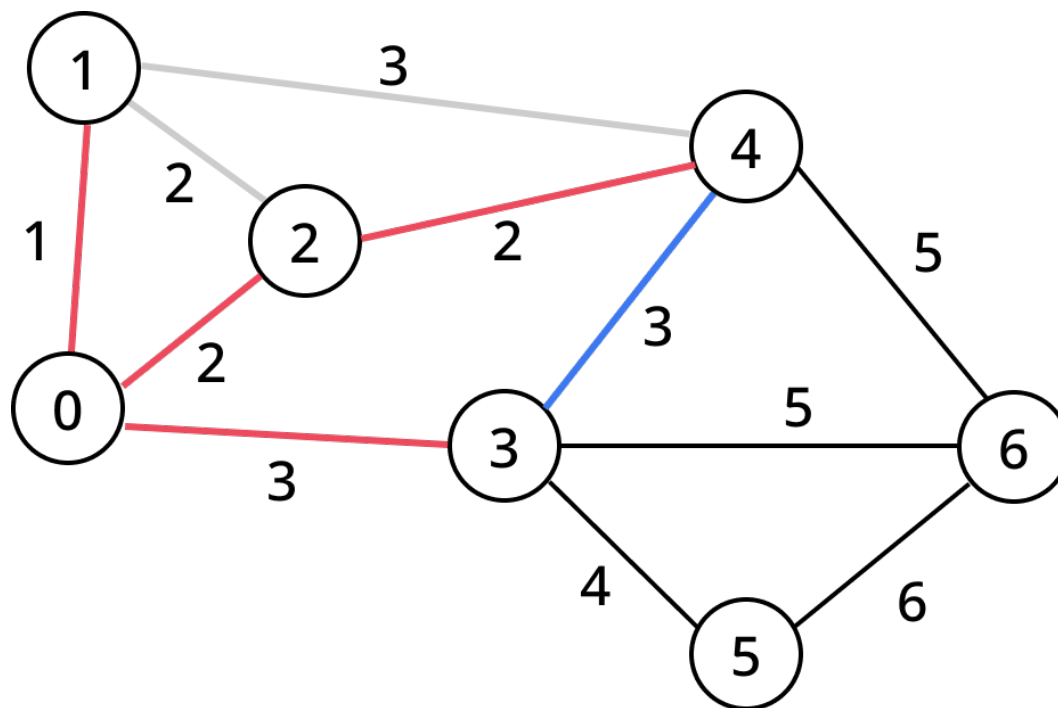
간선을 추가해도 여전히 스패닝 트리를 만족하는 간선을 고른다.



크루스칼 알고리즘

- 가중치가 작은 간선부터 고를 수 있으면 고른다.

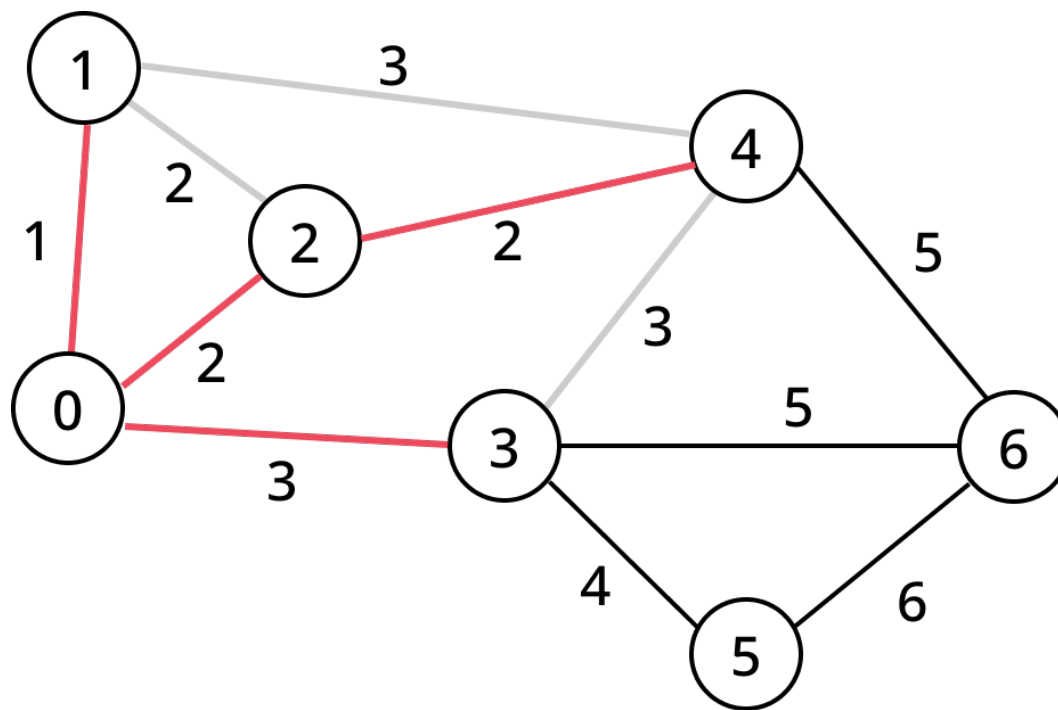
간선을 추가해도 여전히 스패닝 트리를 만족하는 간선을 고른다.



크루스칼 알고리즘

- 가중치가 작은 간선부터 고를 수 있으면 고른다.

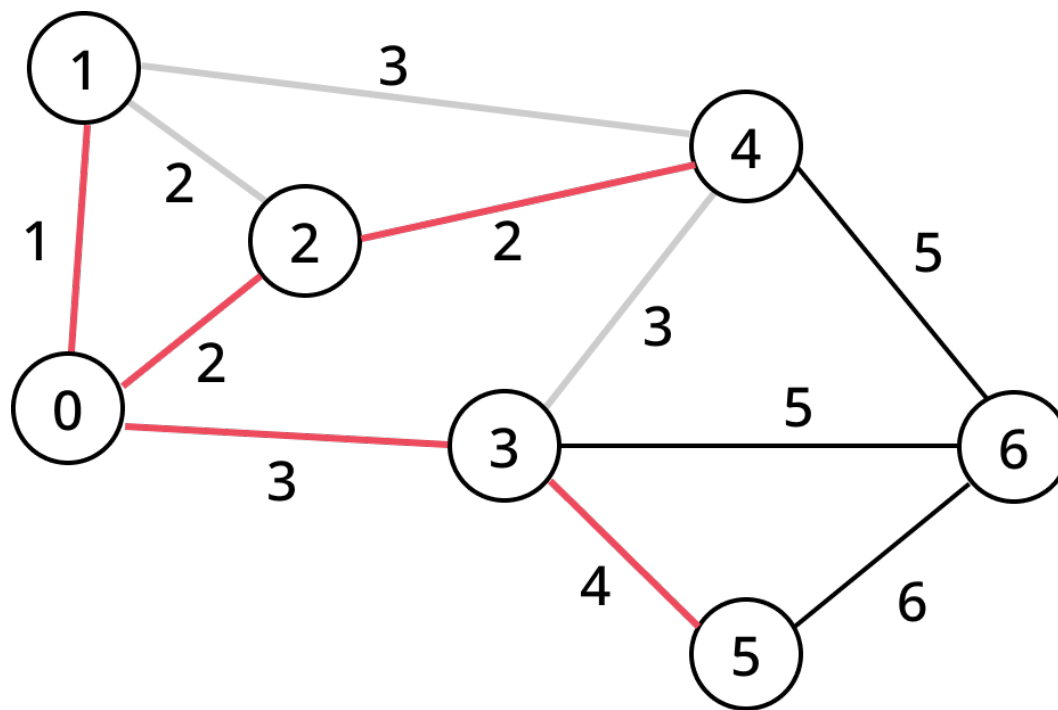
간선을 추가해도 여전히 스패닝 트리를 만족하는 간선을 고른다.



크루스칼 알고리즘

- 가중치가 작은 간선부터 고를 수 있으면 고른다.

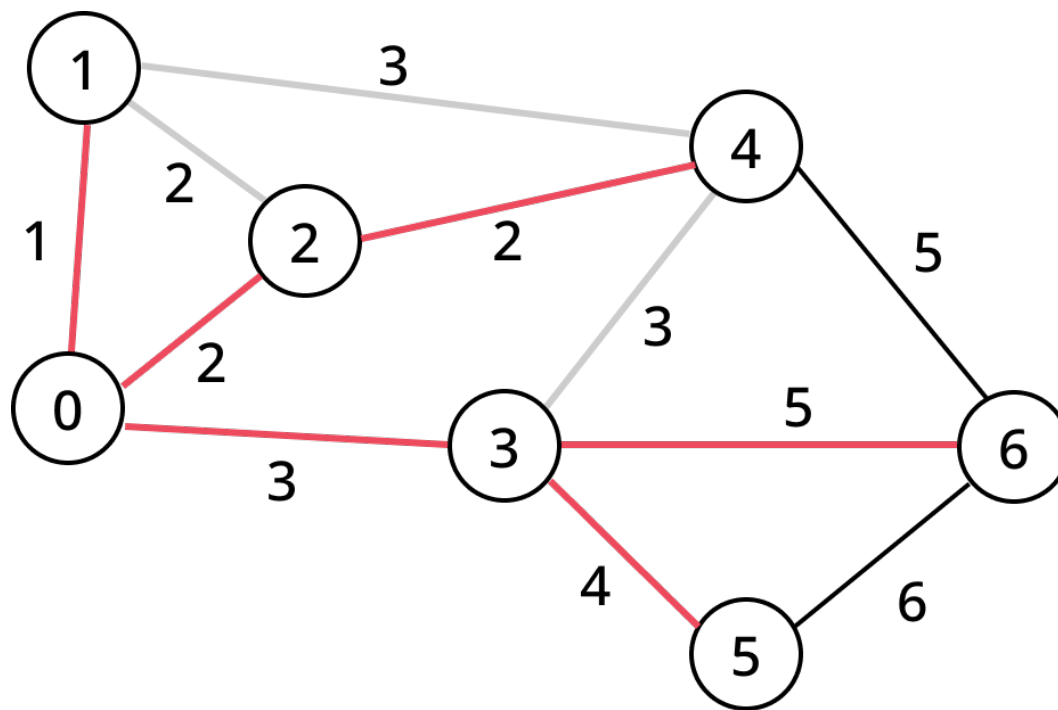
간선을 추가해도 여전히 스패닝 트리를 만족하는 간선을 고른다.



크루스칼 알고리즘

- 가중치가 작은 간선부터 고를 수 있으면 고른다.

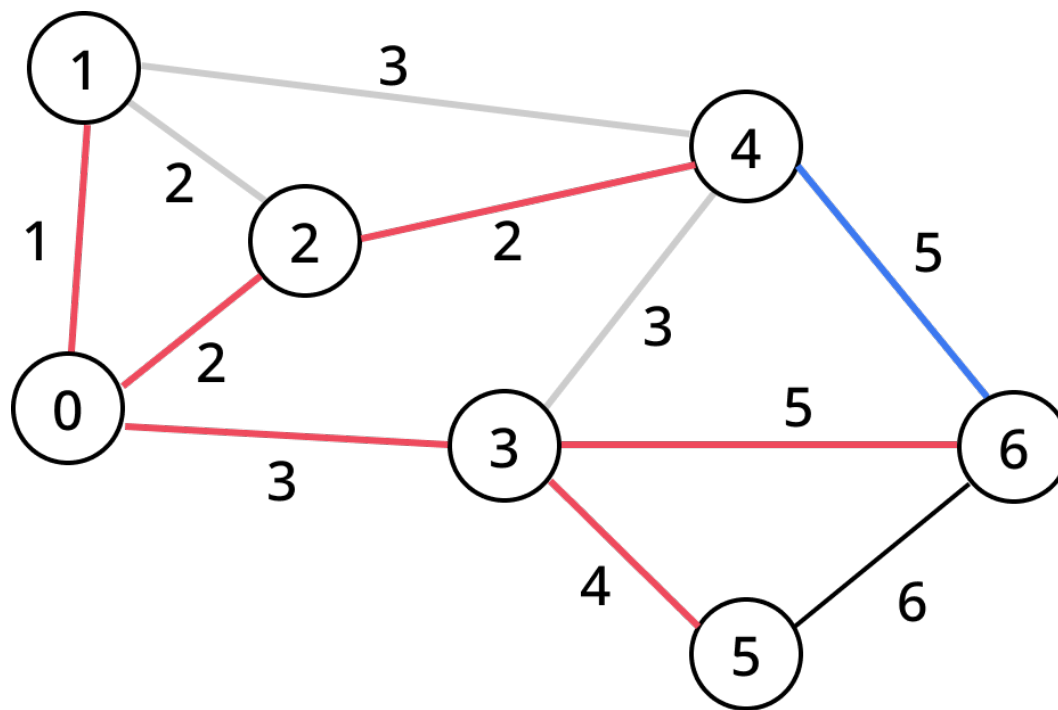
간선을 추가해도 여전히 스패닝 트리를 만족하는 간선을 고른다.



크루스칼 알고리즘

- 가중치가 작은 간선부터 고를 수 있으면 고른다.

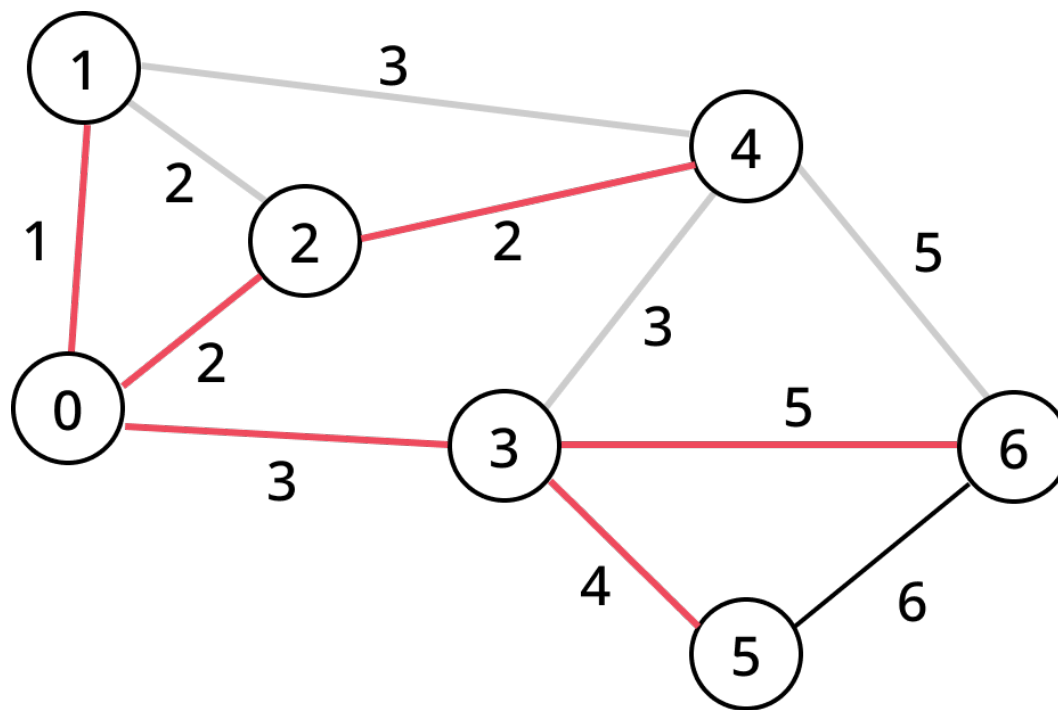
간선을 추가해도 여전히 스패닝 트리를 만족하는 간선을 고른다.



크루스칼 알고리즘

- 가중치가 작은 간선부터 고를 수 있으면 고른다.

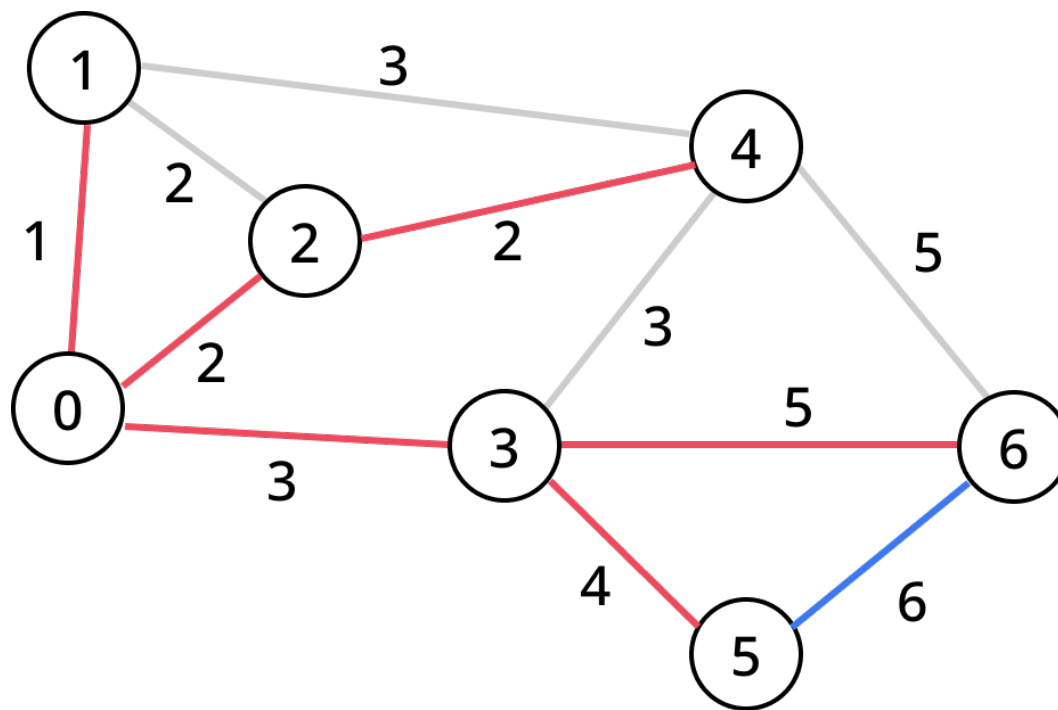
간선을 추가해도 여전히 스패닝 트리를 만족하는 간선을 고른다.



크루스칼 알고리즘

- 가중치가 작은 간선부터 고를 수 있으면 고른다.

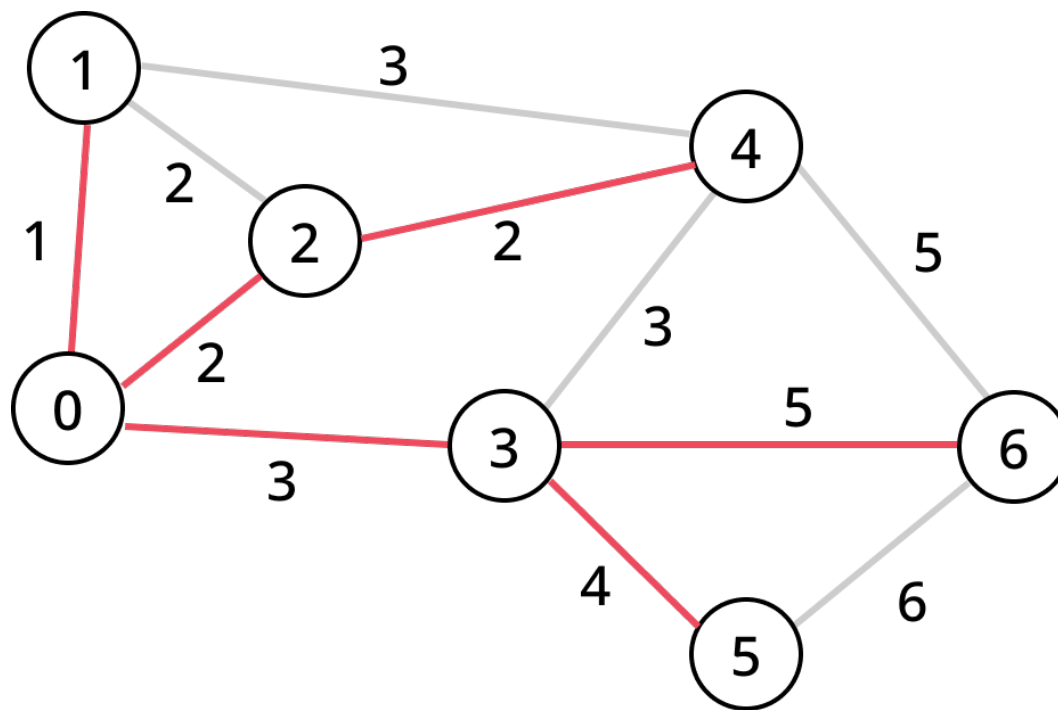
간선을 추가해도 여전히 스패닝 트리를 만족하는 간선을 고른다.



크루스칼 알고리즘

- 가중치가 작은 간선부터 고를 수 있으면 고른다.

간선을 추가해도 여전히 스패닝 트리를 만족하는 간선을 고른다.



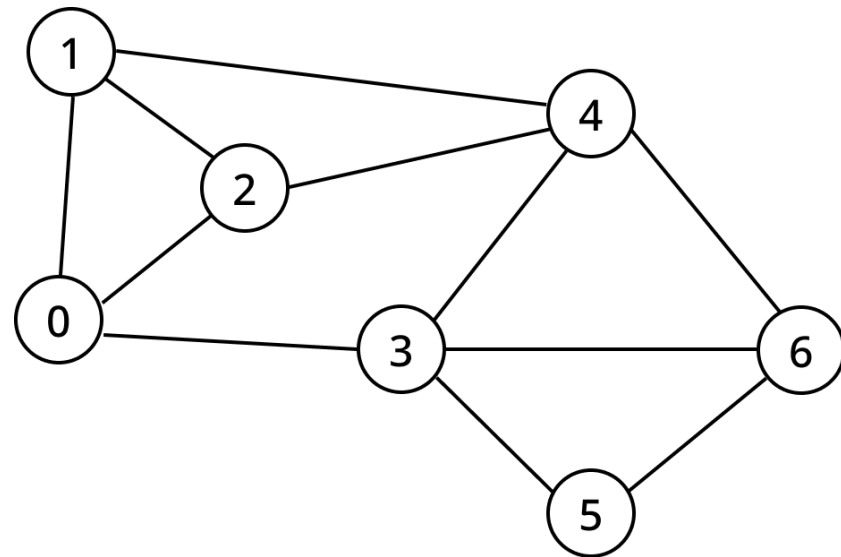
크루스칼 알고리즘

• 크루스칼 알고리즘의 동작 과정

1. 간선의 목록을 가중치 순서대로 오름차순 정렬한다.
2. 간선을 순회하며 이미 선택한 간선과 사이클이 생기지 않으면 스패닝 트리에 추가한다.

• 2번 과정을 어떻게 할 수 있을까?

1. 트리에 간선을 추가한 뒤 DFS로 역방향 간선이 있는지 판별한다.
→ 각 간선마다 한 번씩 DFS를 수행해야하므로 $O(E^2)$ 의 시간이 걸린다.



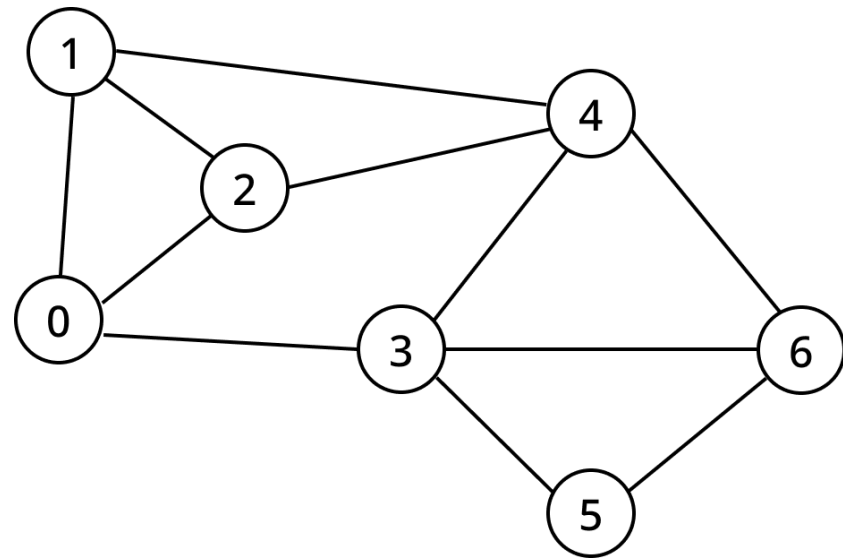
크루스칼 알고리즘

• 크루스칼 알고리즘의 동작 과정

1. 간선의 목록을 가중치 순서대로 오름차순 정렬한다.
2. 간선을 순회하며 이미 선택한 간선과 사이클이 생기지 않으면 스패닝 트리에 추가한다.

• 2번 과정을 어떻게 할 수 있을까?

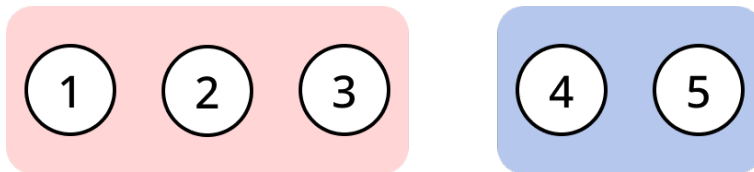
1. 트리에 간선을 추가한 뒤 DFS로 역방향 간선이 있는지 판별한다.
→ 각 간선마다 한 번씩 DFS를 수행해야하므로 $O(E^2)$ 의 시간이 걸린다.
2. 유니온 파인드 자료구조를 사용한다.



참고

- 서로소 집합

서로 공통원소가 없는 두 집합을 의미한다.



- 유니온 파인드

서로소 집합을 표현하는 자료구조

- 유니온 파인드 연산

- 초기화 : n 개의 원소가 각각의 집합에 포함되어 있도록 초기화한다.
- 합치기(union) : 두 원소 a, b 가 주어질 때 이들이 속한 두 집합을 하나로 합친다.
- 찾기(find) : 어떤 원소 a 가 주어질 때 이 원소가 속한 집합을 반환한다.

시간복잡도

• 크루스칼 알고리즘의 시간복잡도

1. 간선의 목록을 가중치 순서대로 오름차순 정렬한다.
2. 간선을 순회하며 이미 선택한 간선과 사이클이 생기지 않으면 스패닝 트리에 추가한다.

총 시간복잡도는 $O(E \log E)$

```
#include <bits/stdc++.h>
using namespace std;

struct dsu {
    vector<int> parent;
    void init(int n){
        parent.resize(n + 1);
        iota(parent.begin(), parent.end(), 0);
    }
    int find(int x){
        return parent[x] = (parent[x] == x ? x : find(parent[x]));
    }
    bool unite(int u, int v){
        u = find(u);
        v = find(v);
        if(u < v) swap(u, v);
        if(u == v) return false;
        parent[v] = u; return true;
    }
} dsu;

struct Edge {
    int from, to, weight;
    bool operator< (const Edge& e) const {
        return weight < e.weight;
    }
};

int main() {
    freopen("input.txt", "r", stdin);
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    int V, E;
    cin >> V >> E;

    vector<Edge> edges;
    for(int i = 0; i < E; i++) {
        int from, to, weight;
        cin >> from >> to >> weight;
        edges.push_back({from, to, weight});
    }
    sort(edges.begin(), edges.end());

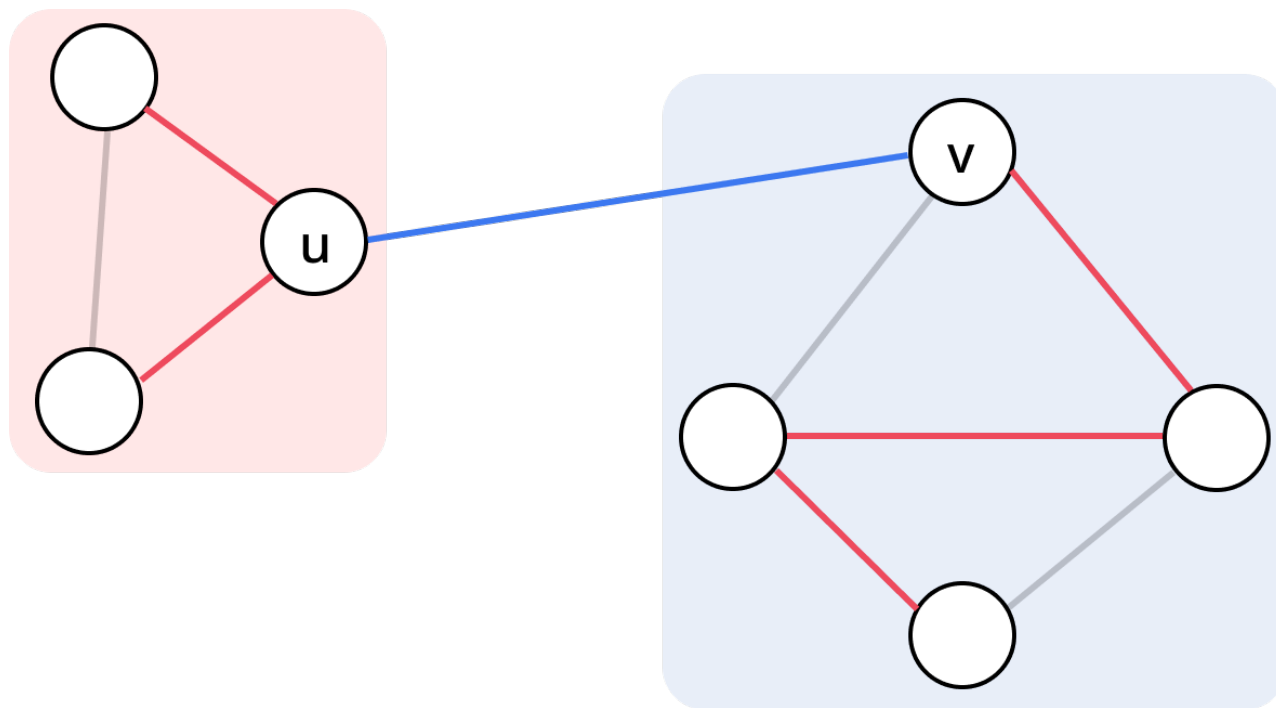
    long long answer = 0;
    dsu.init(V + 1);
    for(int i = 0; i < E; i++) {
        if(dsu.unite(edges[i].from, edges[i].to)) {
            answer += edges[i].weight;
        }
    }

    cout << answer;
    return 0;
}
```

정당성 증명

- 귀류법을 사용하자.

크루스칼 알고리즘이 선택한 간선 (u, v) 가 최소 스패닝 트리에 포함되지 않는다고 가정하자.



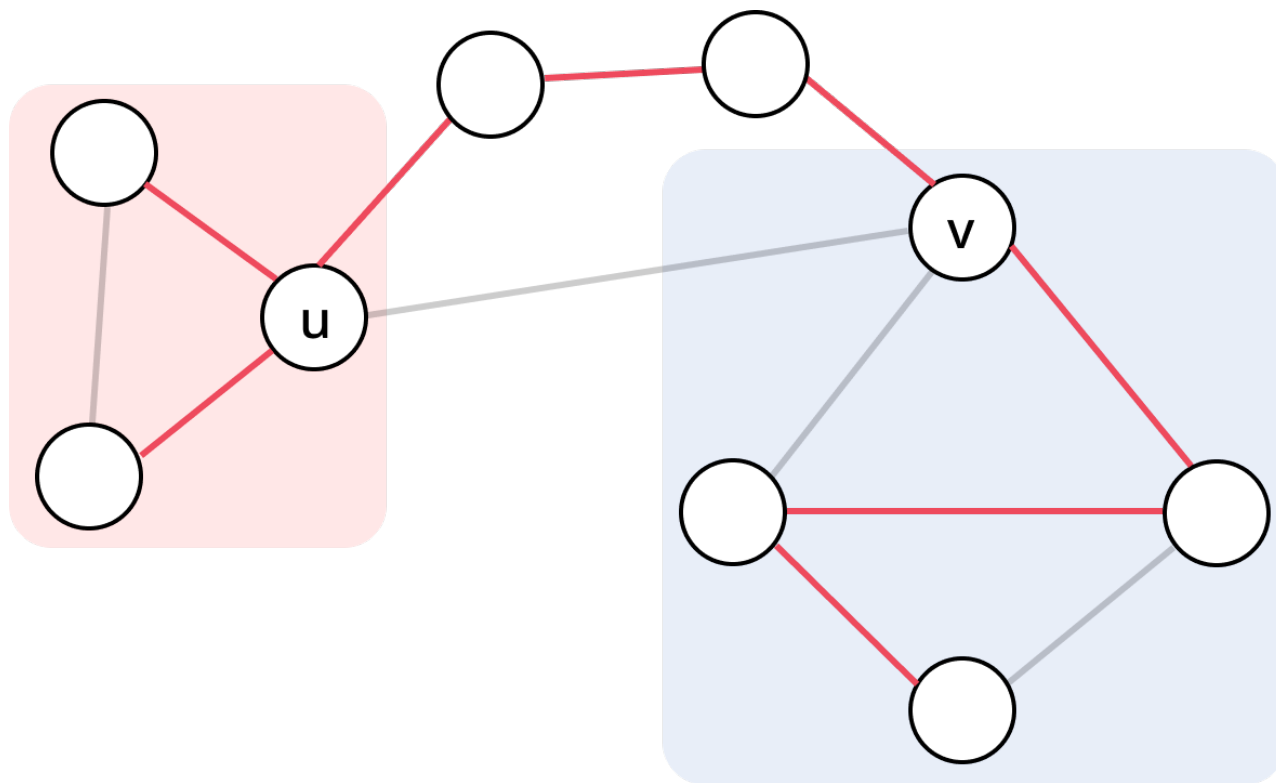
정당성 증명

- 귀류법을 사용하자.

크루스칼 알고리즘이 선택한 간선 (u, v) 가 최소 스패닝 트리에 포함되지 않는다고 가정하자.

진짜 스패닝 트리에서는 u, v 가 최소 스패닝 트리 상에서 다른 경로로 연결되어 있을 것이다.

이 때 이 경로의 간선들은 (u, v) 의 가중치보다 크거나 같다.



정당성 증명

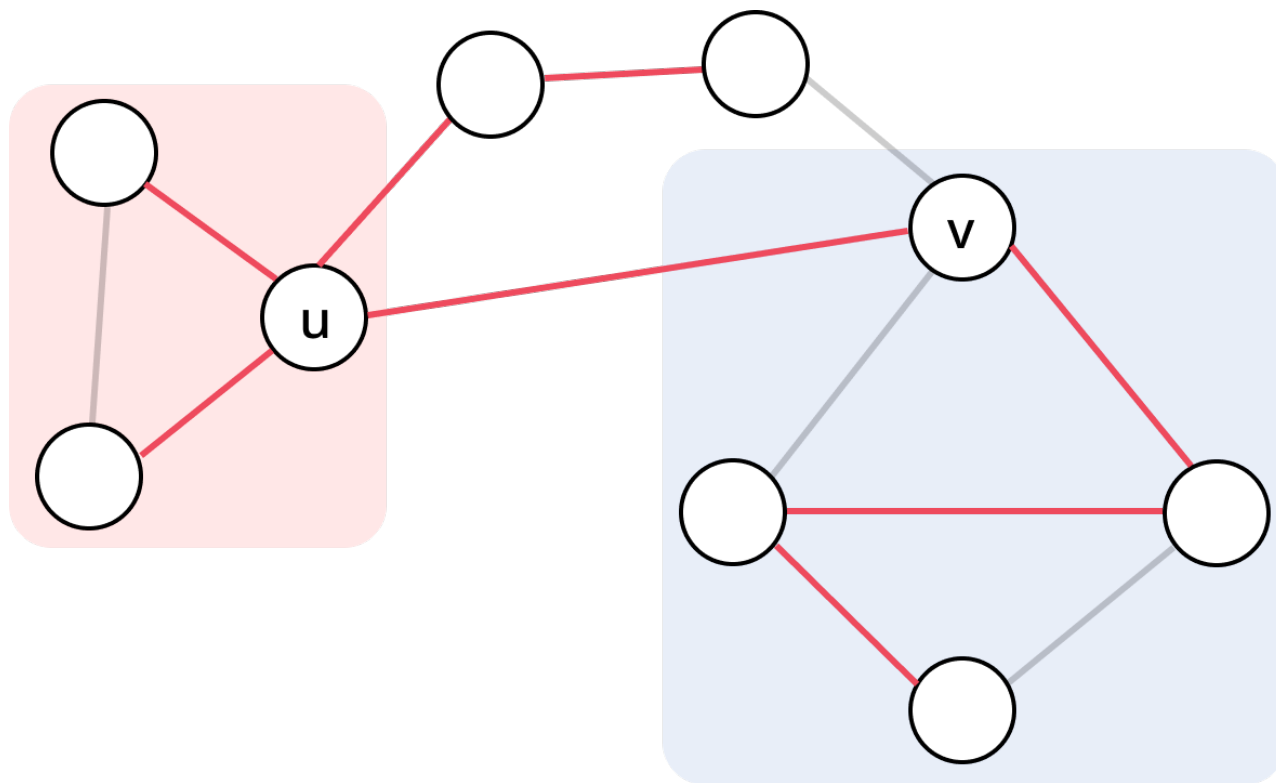
- 귀류법을 사용하자.

크루스칼 알고리즘이 선택한 간선 (u, v) 가 최소 스패닝 트리에 포함되지 않는다고 가정하자.

진짜 스패닝 트리에서는 u, v 가 최소 스패닝 트리 상에서 다른 경로로 연결되어 있을 것이다.

이 때 이 경로의 간선들은 (u, v) 의 가중치보다 크거나 같다.

따라서 이 경로상에서 (u, v) 이상의 가중치를 갖는 간선을 하나꼴라 지워버리고 (u, v) 를 이으면 길이가 더 짧아지므로 모순이다.



정당성 증명

- 귀류법을 사용하자.

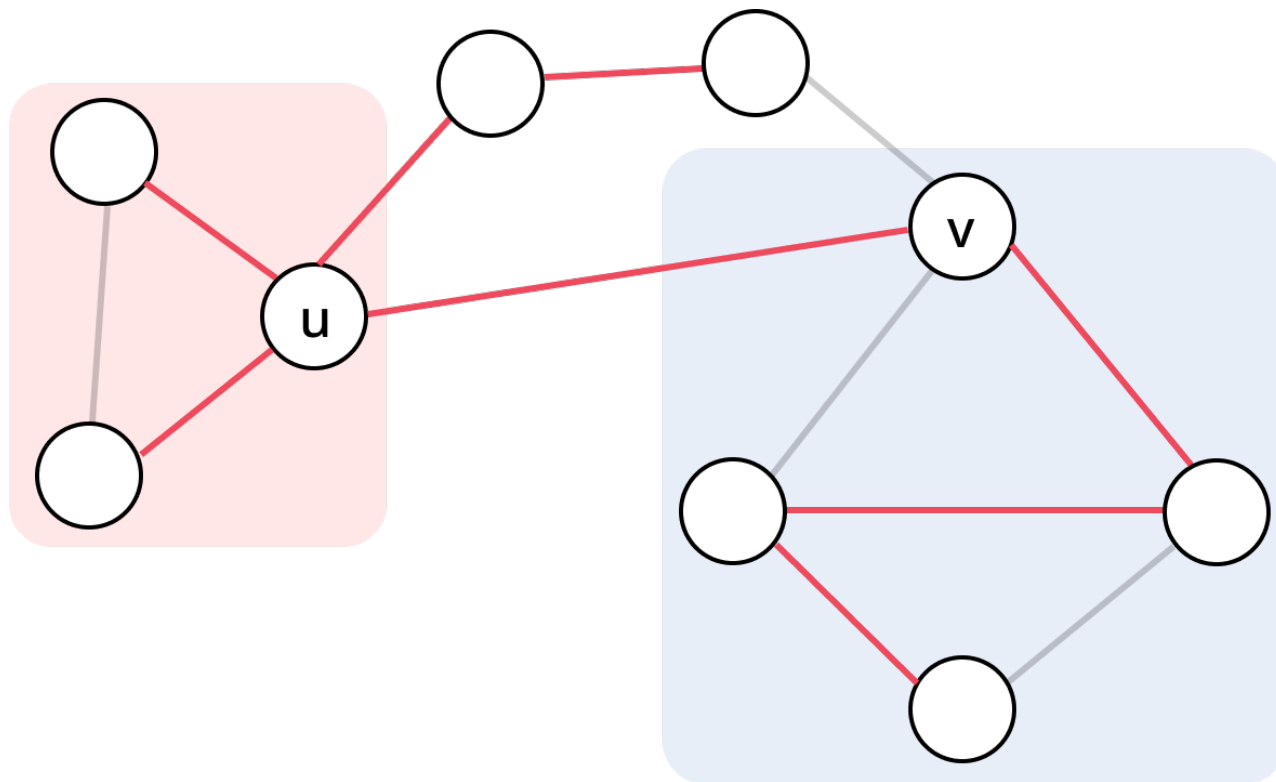
크루스칼 알고리즘이 선택한 간선 (u, v) 가 최소 스패닝 트리에 포함되지 않는다고 가정하자.


진짜 스패닝 트리에서는 u, v 가 최소 스패닝 트리 상에서 다른 경로로 연결되어 있을 것이다.

이 때 이 경로의 간선들은 (u, v) 의 가중치보다 크거나 같다.

따라서 이 경로상에서 (u, v) 이상의 가중치를 갖는 간선을 하나꼴라 지워버리고 (u, v) 를 이으면 길이가 더 짧아지므로 모순이다.

즉 (u, v) 를 선택하여도 남은 간선들을 잘 선택하면 항상 최소 스패닝 트리를 완성할 수 있다.





Q & A