



ALOHA

#5주차

Disjoint Set & MST

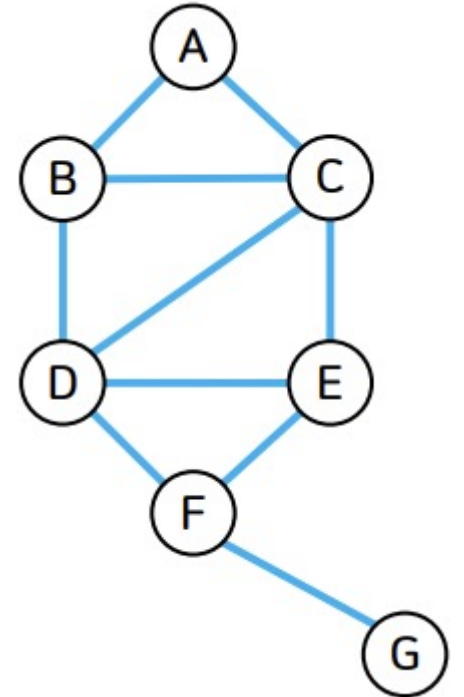
#CH.0

그래프와 트리



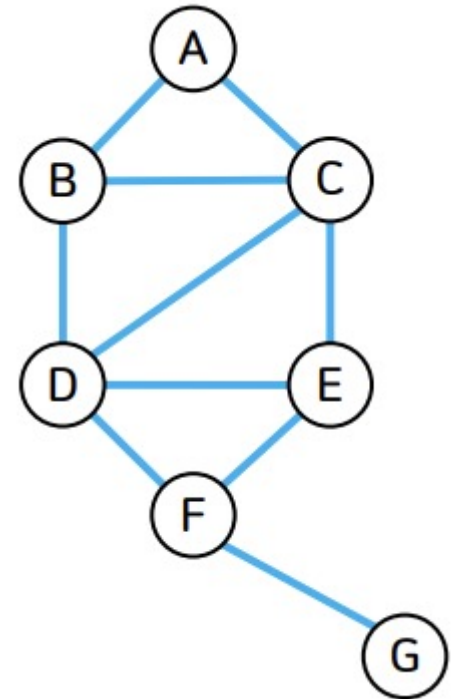
그래프란?

- 일부 객체들의 쌍들이 서로 연관된 객체의 집합을 이루는 구조
 >>> 두 개체의 관계를 나타낼 수 있는 구조
- 일련의 꼭짓점들과 그 사이를 잇는 변들로 구성된 조합론적 구조
 >>> 각 개체는 정점(꼭짓점), 개체 간의 관계는 간선(변)으로 나타냄
- 그래프가 가지는 특성에 따라 가중치 그래프, 단방향 그래프, 양방향 그래프 등등이 있다.



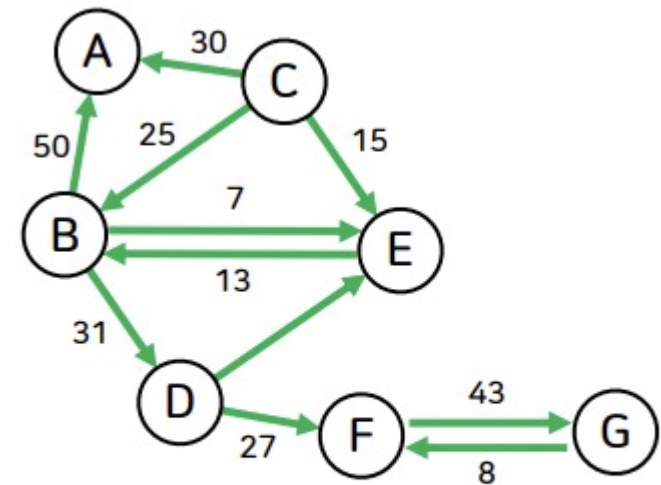
상식으로 알아두는 그래프 용어

- 정점 == 노드 == vertex
- 간선 == edge
- 가중치 == weight



가중치란?

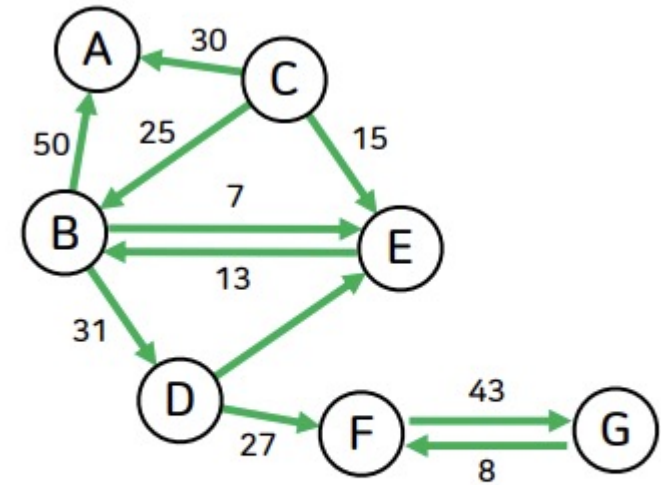
- 그래프의 간선에 붙는 수치
- 간선으로 연결된 정점 간의 관계를 표현하는데 사용한다.
- 가중치가 나타낼 수 있는 관계는 다음 슬라이드를 참고하자.



가중치가 있는 단방향 그래프

가중치의 예시

- 두 지점 간의 거리
ex) B에서 A로 가는 도로의 길이는 50이다.
ex) C에서 B로 가는 도로의 길이는 25이다.
- 이동하는 데 드는 비용
ex) B에서 E로 가는 것은 13만큼의 비용이 든다.
ex) F에서 G로 가는 것은 43만큼 시간이 든다.
- 여타 여러가지 “연결 ” 에 관한 값들
ex) 도로 설치 비용, 이동 시간, ...

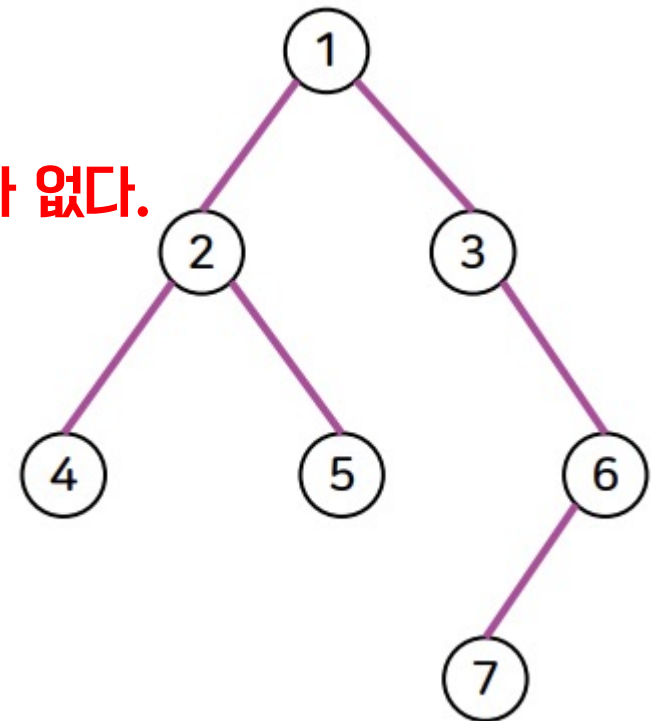


가중치가 있는 단방향 그래프

트리란?

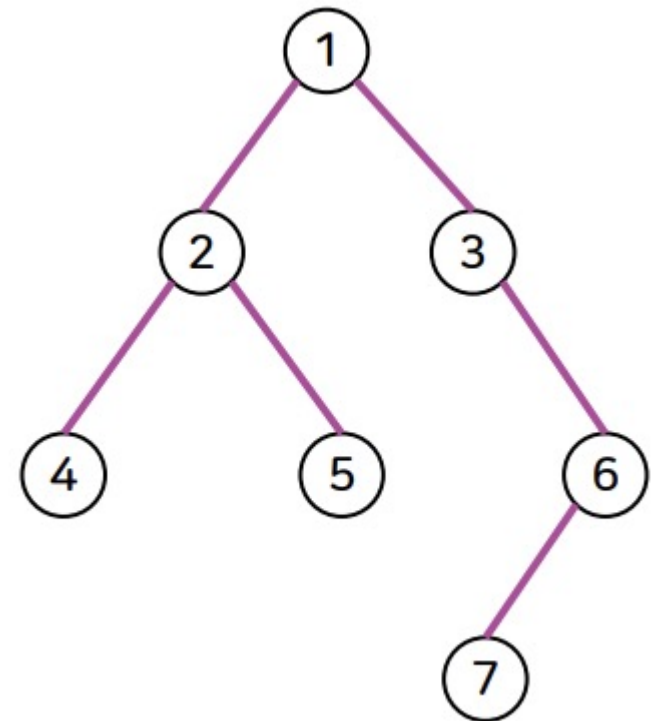
- 사이클이 없는 연결 그래프

* 사이클이 없다? :
어떤 노드에서 시작해서 다시 자기 자신으로 돌아오는 경로가 없다.



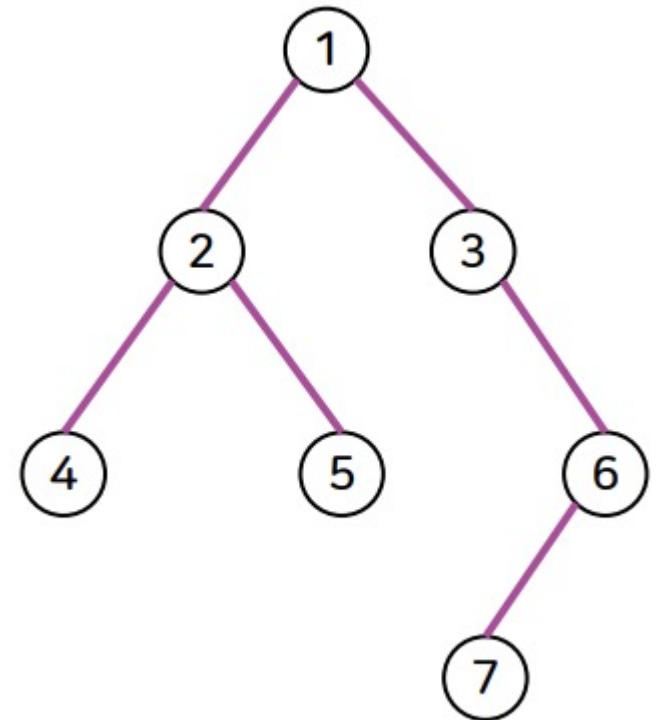
상식으로 알아두는 트리 용어

- 부모 노드와 자식 노드 :
어떤 노드의 상위 노드를 그 노드의 부모 노드라 한다.
이때 어떤 노드는 부모 노드의 자식 노드가 된다.
ex) 노드 4는 노드 2의 자식 노드이다.
노드 3은 노드 6의 부모 노드이다.
- Root 노드 :
부모 노드가 없는 노드, 최상위 노드
ex) 오른쪽 트리의 root 노드는 1이다.
- Leaf 노드 :
자식이 없는 노드
ex) 오른쪽 트리의 leaf 노드는 4, 5, 7 이다.



트리의 특징

- 모든 트리는 하나의 root 노드를 가진다.
- 모든 노드는 root 노드와 연결된 유일한 경로를 가진다.
- Root 노드 이외의 모든 노드는 부모 노드를 가진다.
- 트리는 (노드 개수) - 1개의 간선을 가진다.
루트 노드 이외의 모든 노드가 부모 노드를 갖고,
부모 노드와 자식 노드는 간선으로 연결 되어있기 때문.



#CH.1

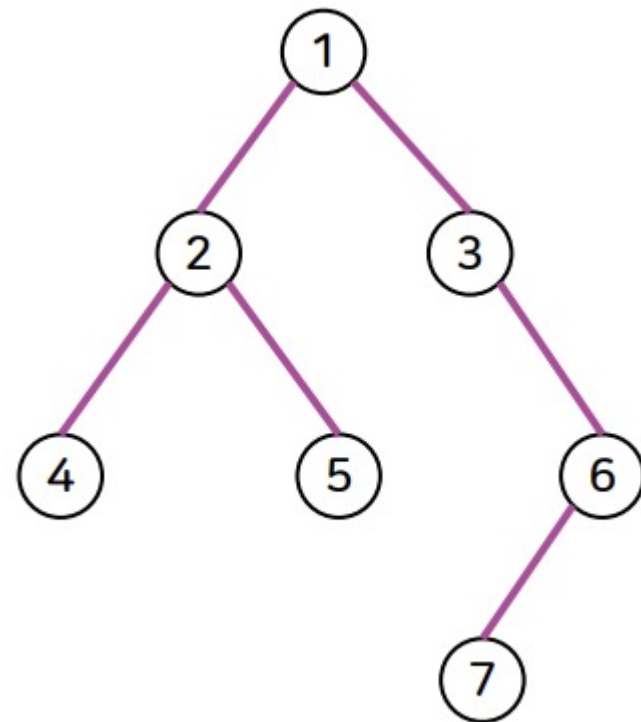
Disjoint Set



시작하기 전에 ...

Disjoint Set 구현을 위해 트리의 특징을 되짚어 보자.

- 모든 트리는 하나의 root 노드를 가진다.
- 모든 노드는 root 노드와 연결된 유일한 경로를 가진다.
- Root 노드 이외의 모든 노드는 부모 노드를 가진다.

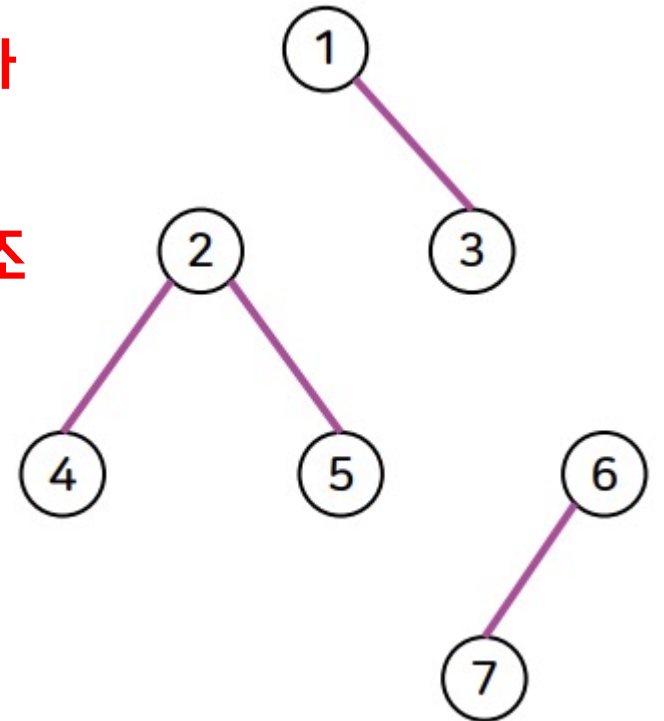


Disjoint Set?

- 서로소 부분 집합을 관리하는 자료구조
 - *서로소 부분 집합 : 어떤 집합의 부분 집합 중 겹치는 원소가 없는 부분집합들

>>> 그룹을 묶고, 그 그룹을 관리하기 위해 사용하는 자료구조

- 기본적으로 두가지 연산을 지원한다.
 1. UNION – 주어진 두 개의 집합을 하나로 합친다.
 2. FIND – 주어진 원소가 어떤 집합에 속해 있는지 찾는다.
(구현은 그 집합을 대표하는 원소를 정해서 그 값을 리턴)



구현법 - IDEA

- 주로 Disjoint Set은 Tree의 형태로 구현된다.
- 트리의 root node를 집합의 대표 원소로 취급한다.
- 즉, 내가 소속된 트리의 root를 알아낼 수만 있다면?
Disjoint Set의 FIND 함수를 구현했다고 할 수 있다.
- 또한, 찾아낸 두 트리를 하나로 이을 수만 있다면?
Disjoint Set의 UNION 함수를 구현했다고 할 수 있다.

구현법 – IMPLEMENTATION

FIND 부터 구현해보자.

- FIND는 주어진 노드가 어느 집합에 속하는지 찾아야 한다.

여기서 우리는 트리의 성질 몇가지를 생각해야 한다.

- 모든 노드는 root 노드와 연결된 유일한 경로를 가진다.
- Root 노드 이외의 모든 노드는 단 하나의 부모 노드를 가진다.

이 성질을 잘 생각해보면, 주어진 노드의 부모 노드를 찾고,
그 부모 노드의 부모 노드를 찾고 ..., 이런 식으로 반복하다 보면
결국 root 까지 도달할 수 있다는 것을 알 수 있다.

```
4  int parent[SIZE];
5
6  int find(int node){
7      if(node == parent[node]){
8          return node;
9      }
10     return find(parent[node]);
11 }
12
```

처음에 parent[i] = i 이도록 해야 한다.

Find 함수의 구현

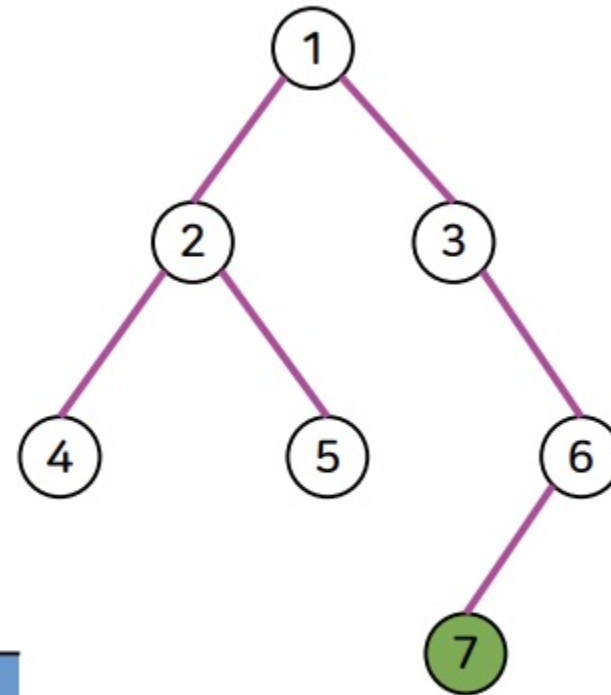
만약 자신이 루트 노드라면 (자신의 부모가 자신) 현재 노드를 리턴.

아니라면, **내 부모 노드의 루트 노드를 찾아 리턴.** (find(parent[node]))

어떻게 동작할까? - find

현재 노드
7

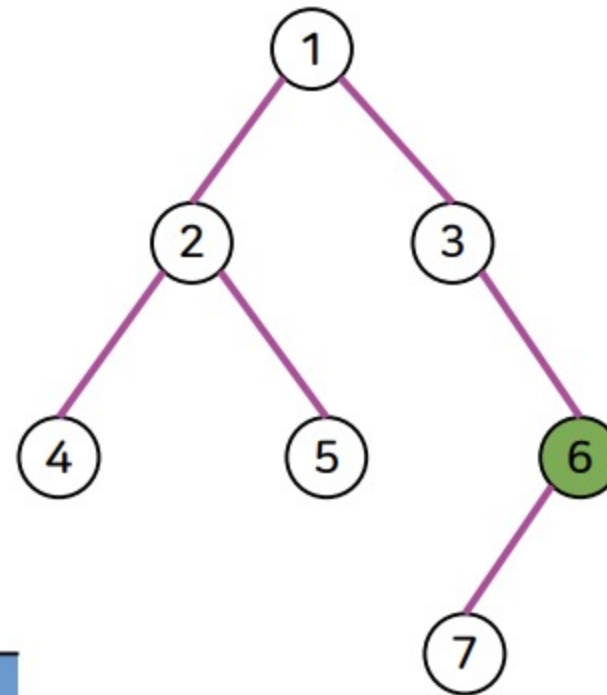
노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	3	6



어떻게 동작할까? - find

현재 노드
6

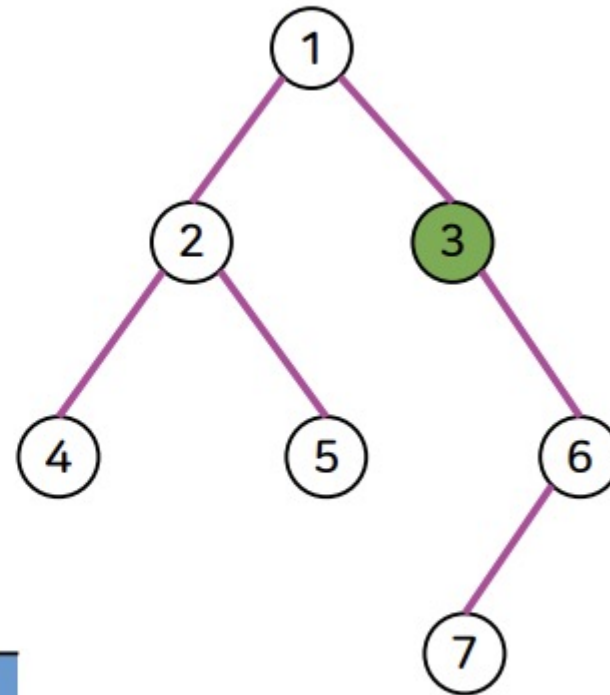
노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	3	6



어떻게 동작할까? - find

현재 노드
3

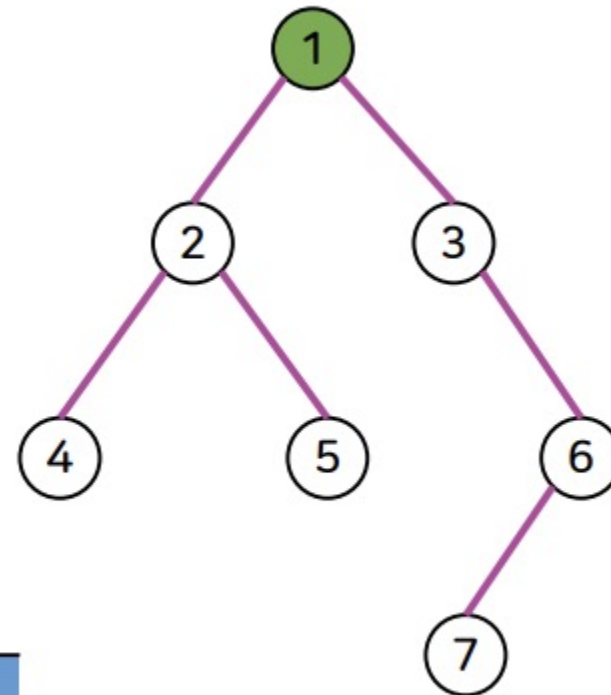
노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	3	6



어떻게 동작할까? - find

현재 노드
1

노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	3	6





이게 최선일까요?

구현법 – IMPLEMENTATION

- 앞에서 우리가 구현한 FIND는 인자로 주어진 노드의 깊이만큼 재귀호출이 일어난다. 즉, root에서 멀어지면 멀어질 수록, FIND 연산에 걸리는 시간도 길어지게 된다.
- 만약 노드의 깊이가 m 인 노드를 가지고 FIND를 n 번 사용한다면 어떻게 될까?
당연히 시간복잡도는 $O(nm)$ 이 될 것이다.
 $n = 1e5$, $m=1e3$ 정도만 되어도 $nm = 1e8$ 이므로 너무 오래 걸린다.

거기다 이 다음에 구현할 UNION 또한 FIND를 사용하기 때문에 FIND가 오래 걸릴 경우, 논리를 맞게 구성해도 문제에서 시간초과가 나올 가능성이 높다.

구현법 – IMPLEMENTATION

- 그럼 어떻게 개선할까?
- 사실 우리가 궁금한 건, “나는 어느 집합에 소속되어 있나?” 이지, “내 부모는 누구인가?”가 아니다. 그렇다면 굳이 트리의 형태를 그대로 유지할 필요는 없다.
“같은 집합에 있다” 라는 조건만 유지한다면, 트리의 구조는 얼마든지 변형해도 무방하다.
- 여기까지 이해했다면 어느 정도 답이 보일 것이다.
루트 이외의 모든 노드가 루트를 부모로 가질 수 있도록 하면 어떨까?

```
4 int parent[SIZE];  
5  
6 int find(int node){  
7     if(node == parent[node]){  
8         return node;  
9     }  
10    return parent[node] = find(parent[node]);  
11 }  
12
```

여기가 바뀌었어요

Find 함수의 구현

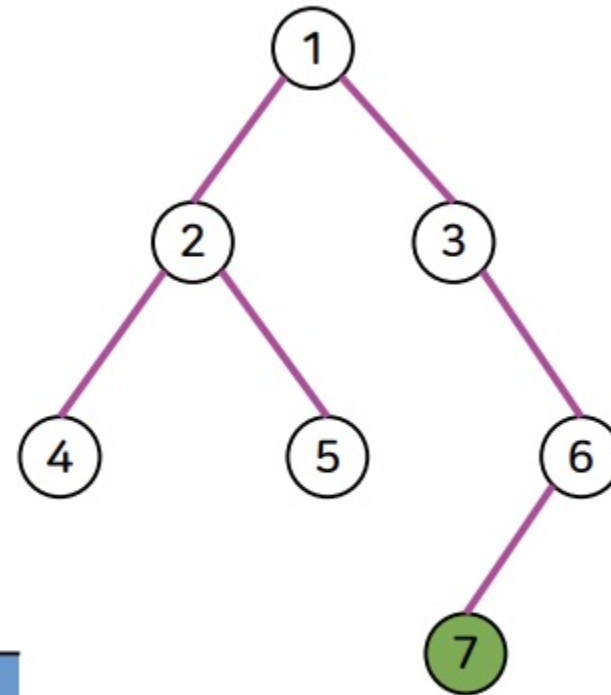
나의 부모를 내 부모의 부모로 바꾸도록 하자.

어차피 모두 같은 집합 소속이므로 문제는 없다.

어떻게 동작할까? - find

현재 노드
7

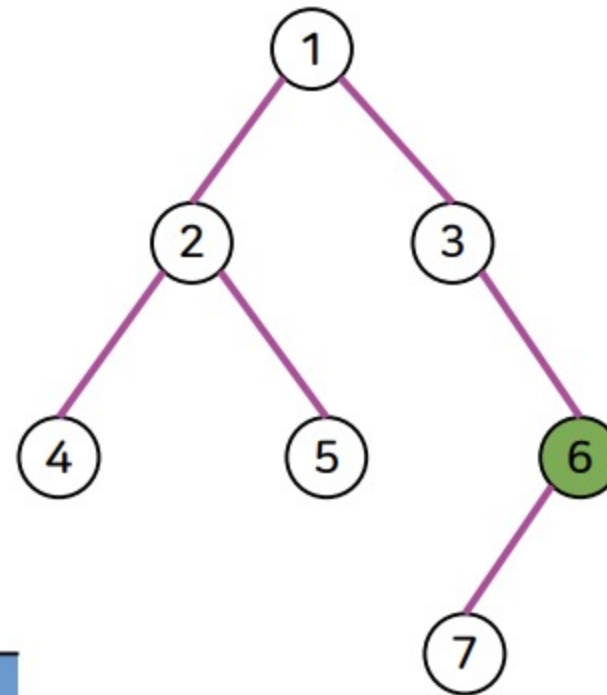
노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	3	6



어떻게 동작할까? - find

현재 노드
6

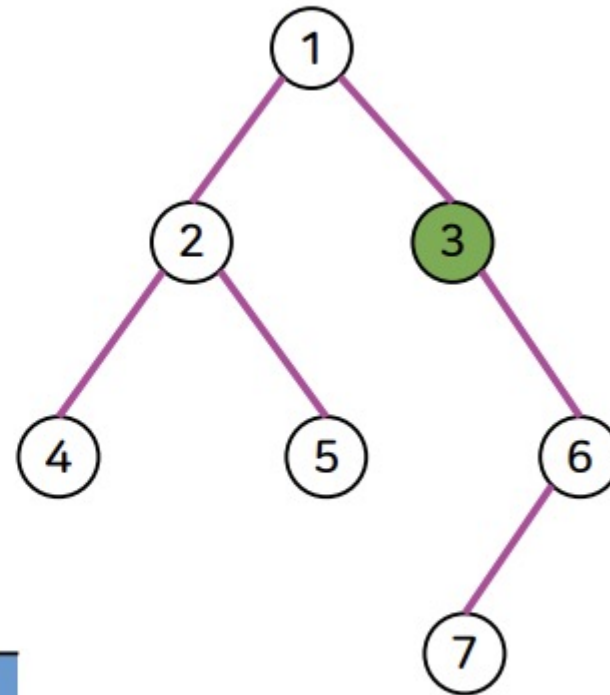
노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	3	6



어떻게 동작할까? - find

현재 노드
3

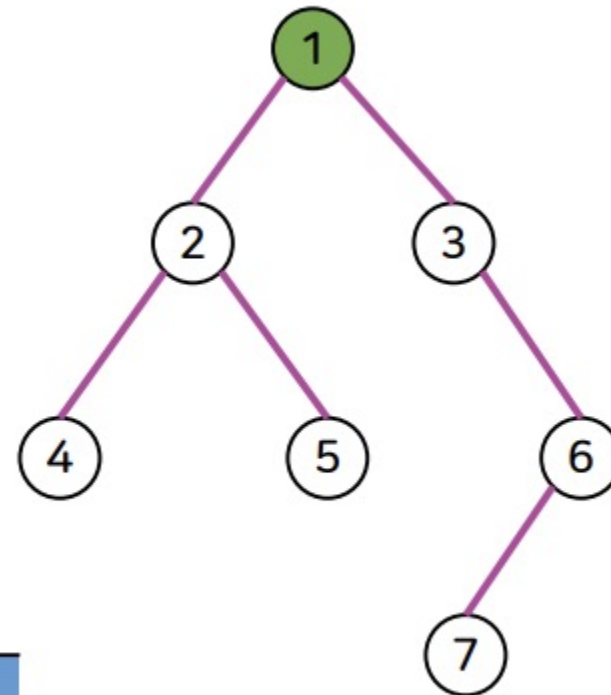
노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	3	6



어떻게 동작할까? - find

현재 노드
1

노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	3	6



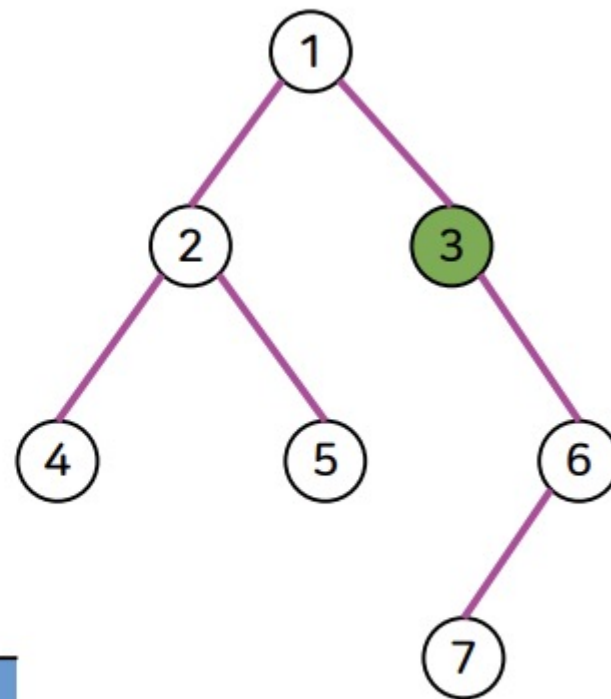


여기까지는 앞의 구현과 같아요.
하지만 재귀 호출이 끝나고 돌아오면서 트리 모양이 변하게 됩니다.

어떻게 동작할까? - find

현재 노드	return 값
3	1

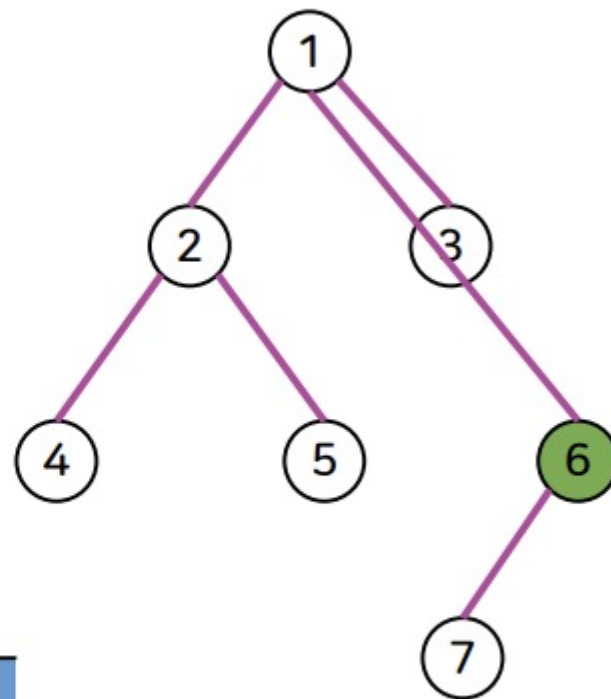
노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	3	6



어떻게 동작할까? - find

현재 노드	return 값
6	1

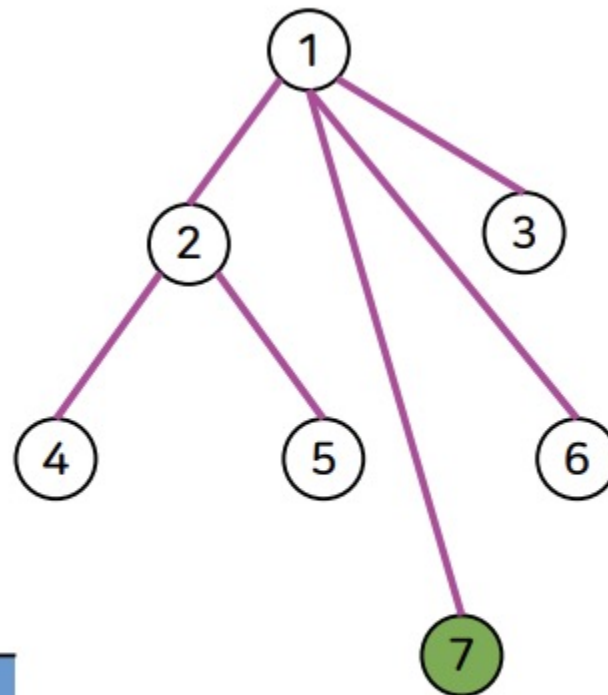
노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	1	6




어떻게 동작할까? - find

현재 노드	return 값
7	1

노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	1	1





이제 `find(6)`, `find(7)`을 다시 하면 한 번 만에 답을 찾을 수 있어요.
이러한 방식을 Path Compression 이라고 합니다.

구현법 – IMPLEMENTATION

- 이제 UNION을 구현해보자. – UNION은 두 집합을 하나로 합치는 것이다.
- 두 집합을 하나로 합치기 위해서는 두 집합의 root 를 같게 해야 한다.
- Root 노드 이외의 노드는 단 하나의 부모 노드를 갖는다.

두 집합을 전달 받았다는 것은 두 트리의 루트를 전달받았다고 해석할 수 있다.
위에서 언급한 트리의 특징을 생각하면 두 루트 노드 모두 부모 노드가 없고
두 노드를 a, b라고 하면 a를 b의 자식 노드로 만들면 두 집합을 합칠 수 있다.

```
13 void uni(int a, int b){  
14     int root_a = find(a);  
15     int root_b = find(b);  
16  
17     parent[root_b] = root_a;  
18 }
```

Union 함수의 구현

두 노드의 루트 노드를 찾고

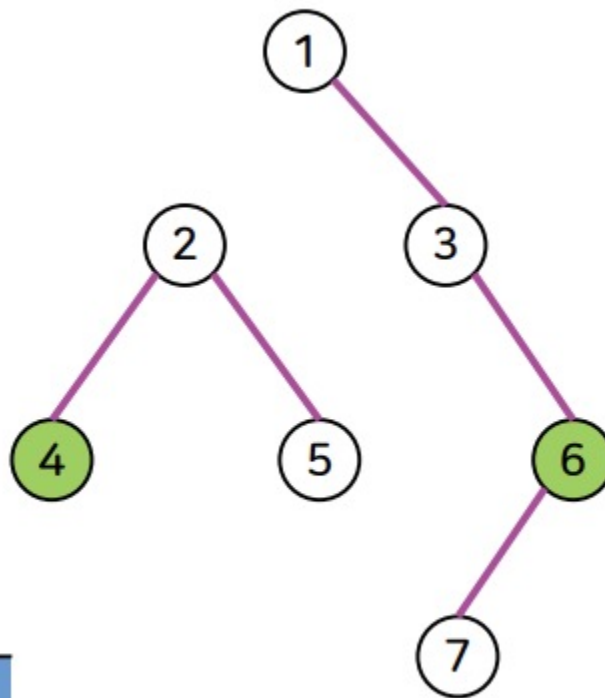
root_b의 부모를 root_a 로 설정하는 것으로 두 집합을 합친다.

어떻게 동작할까? - UNION

노드 a	노드 b
6	4

집합 a	집합 b

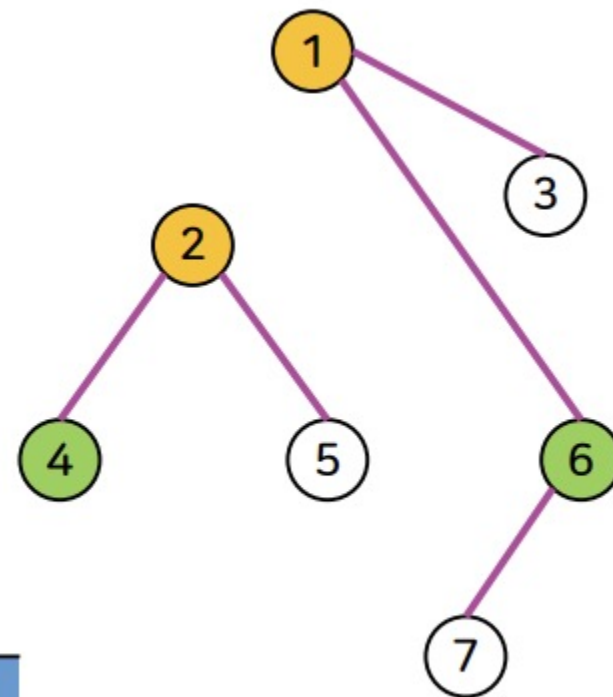
노드	1	2	3	4	5	6	7
부모	1	2	1	2	2	3	6



어떻게 동작할까? - UNION



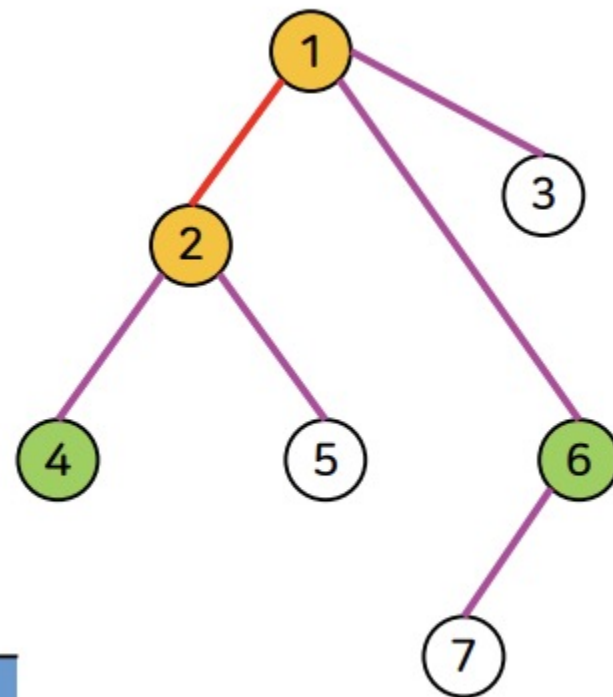
노드	1	2	3	4	5	6	7
부모	1	2	1	2	2	1	6



어떻게 동작할까? - UNION



노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	1	6



풀어볼까유



#1717
집합의 표현

Disjoint Set 연습

풀어볼까유



#1976
여행 가자

Disjoint Set 연습

풀어볼까유



#18116
로봇 조립

Disjoint Set 연습

풀어볼까유



#14595

동방 프로젝트(Large)

Disjoint Set 연습

#1717 – 집합의 표현

전형적인 Disjoint Set 문제이다.

입력 형식에 따라 아래와 같은 코드를 짜면 된다.

0 a b : a 와 b를 UNION 한다.

1 a b : a 와 b가 속한 집합을 FIND를 통해 찾고, 그 둘이 같은지 비교한다.

#1976 – 여행 가자

역시 전형적인 Disjoint Set 문제이다.

주어진 인접행렬을 통해 연결된 도시를 모두 UNION한다.

그리고 여행 계획에 포함된 도시들이 모두 같은 집합에 속하는지 FIND를 통해 판단한다.

모든 같은 집합에 속하면 가능한 계획이고, 하나라도 다른 집합에 속하면 불가능한 계획이다.

#18116 – 로봇 조립

약간의 응용이 필요한 문제이다.

앞의 문제들이 “내가 속한 집합 ” 을 찾는 문제라면, 이번엔 “내가 속한 집합의 크기 ” 를 찾는다.

나이브하게 부품을 일일이 다 찾아보면 시간 초과가 날 것이다.

UNION을 할 때, 새로운 집합의 크기는 두 집합의 크기 합과 같다. 이걸 구현해주면 된다.

#18116 – 로봇 조립

단, 처음에 모든 cnt배열의 값을 1로 초기화 해줘야 하는 것에 주의하자.

```
16 void uni(int a, int b)
17 {
18     int root_a = find(a);
19     int root_b = find(b);
20
21     if (root_a != root_b)
22     {
23         parent[root_b] = root_a;
24         cnt[root_a] = cnt[root_a] + cnt[root_b];
25     }
26 }
27
```

#14595 – 동방 프로젝트 (large)

조금 특이한 문제이다. – 나이브하게 $(y-x)$ 회 UNION을 하면 시간초과를 받는다.

따라서 UNION을 조금 변형해야 한다.

1. 집합을 대표하는 원소는 무조건 가장 큰 번호가 말도록 한다 : $\text{find}(a) \geq a$ 가 보장된다.
2. $[a,b]$ 의 방을 합친다는 구간내 모든 방 x 에 대해 x 가 포함된 집합의 대표 원소가 모두 $\text{find}(b)$ 가 된다는 의미이다.
3. 모든 방의 대표 원소를 바꿀 필요 없이, 기존 집합의 대표 원소였던 방만 root를 $\text{find}(b)$ 로 바꿔준다. 이러면 반복문의 횟수는 크게 줄어든다.

```
16 void uni(int a, int b)
17 {
18     int root_a = find(a);
19     int root_b = find(b);
20
21     for(int i = root_a; i != root_b; i = find(i+1)){
22         parent[root_b] = root_a;
23         cnt[root_a] += cnt[root_b];
24     }
25 }
```

#14595 동방 프로젝트 (large)

시작 : node_a를 포함하는 집합의 루트

조건 : 현재 방이 root_b가 아닐때 까지

다음 : 옆방을 포함하는 집합의 대표번호

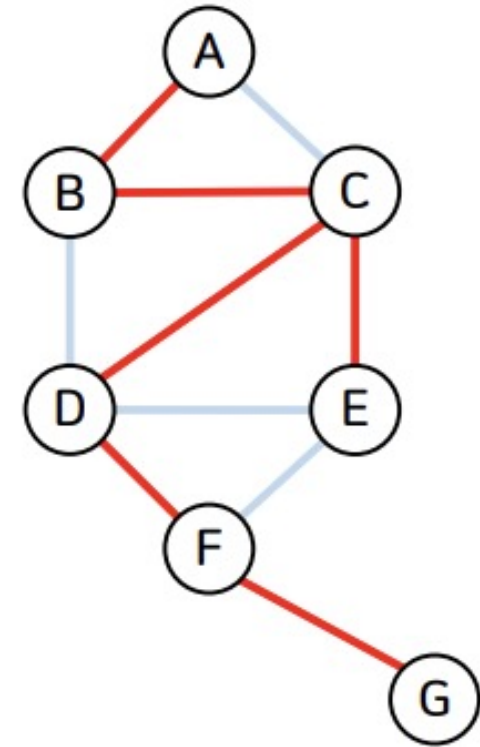
#CH.2

Minimum Spanning Tree 알고리즘을 짜보자!



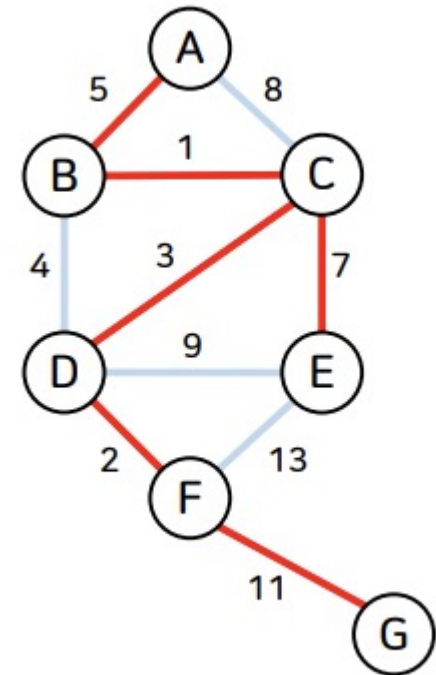
스패닝 트리란?

- 그래프에 포함된 모든 정점을 포함하는 트리
- 옆의 그래프에서 빨간 간선으로 이루어진 부분이 저 그래프의 스패닝 트리이다.
- 당연히 스패닝 트리는 유일하지 않다.



MST란?

- 가중치 그래프에서, 간선의 가중치 합이 가장 작은 스패닝 트리
- 옆의 그래프에서 빨간 간선으로 이루어진 부분이 저 그래프의 MST라고 할 수 있다.
- 이 또한 유일하지 않을 수 있다.



Kruskal 알고리즘

- 주어진 그래프에서 MST를 만드는 알고리즘
- 단계는 다음과 같다.
 1. 그래프의 간선을 가중치 기준으로 오름차순 정렬한다.
 2. 간선의 양 끝 정점이 속한 집합을 찾는다.
 3. 다른 집합이라면 두 집합을 합치고 해당 간선을 MST에 속한 간선으로 취급한다.
 4. 같은 집합이라면 넘어간다.
 5. 모든 간선에 대해 2~4를 반복한다.

Kruskal 알고리즘

- 주어진 그래프에서 MST를 만드는 알고리즘
- 단계는 다음과 같다.
 1. 그래프의 간선을 가중치 기준으로 오름차순 정렬한다.
 2. 간선의 양 끝 정점이 속한 집합을 찾는다. (FIND)
 3. 다른 집합이라면 두 집합을 합치고 (UNION) 해당 간선을 MST에 속한 간선으로 취급한다.
 4. 같은 집합이라면 넘어간다.
 5. 모든 간선에 대해 2~4를 반복한다.

Kruskal 알고리즘도 Disjoint Set을 응용해서 구현한다.

```

29 typedef std::pair<int, std::pair<int, int>> Edge;
30 std::priority_queue<Edge, std::vector<Edge>, std::greater<Edge>> edge;
31
32 int ans;
33
34 int main(void)
35 {
36     while (!edge.empty())
37     {
38         Edge e = edge.top();
39         edge.pop();
40         if (find(e.second.first) != find(e.second.second))
41         {
42             uni(e.second.first, e.second.second);
43             ans += e.first;
44         }
45     }
46
47     return 0;
48 }

```

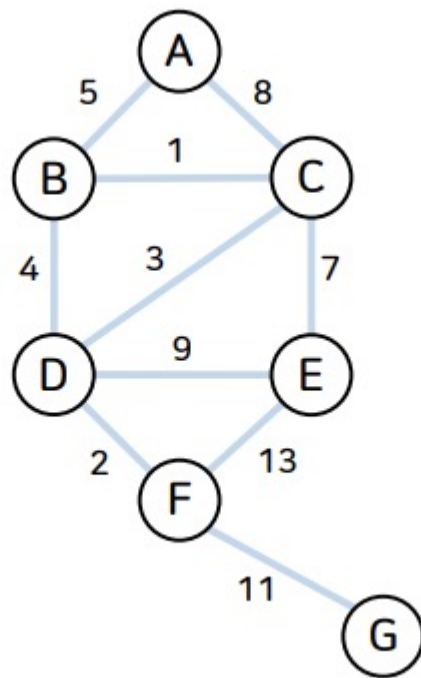
크루스칼 알고리즘

Kruskal 알고리즘을 통해 MST의 가중치 합을 구하는 코드이다.

Find()와 uni()는 위의 정의를 참고하자.

어떻게 동작할까?

가중치 합
0

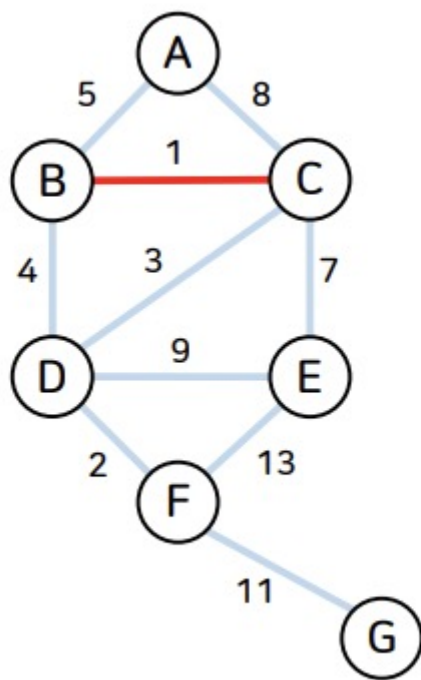


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

B, C는 현재 다른 집합이므로
이 간선을 MST에 포함한다.

가중치 합
0 + 1

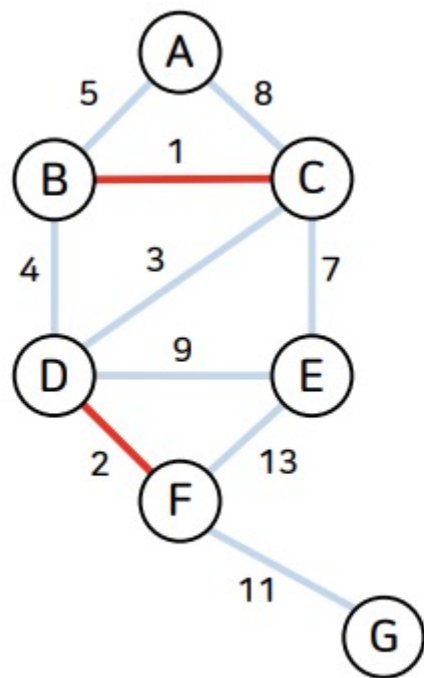


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

D, F는 현재 다른 집합이므로
이 간선을 MST에 포함한다.

가중치 합
1 + 2

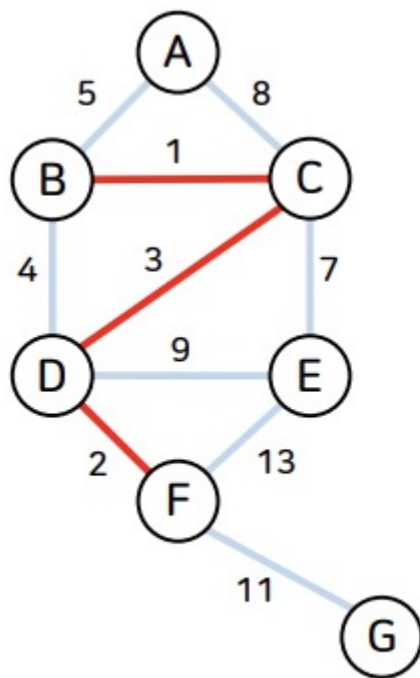


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

C, D는 현재 다른 집합이므로
이 간선을 MST에 포함한다.

가중치 합
3 + 3

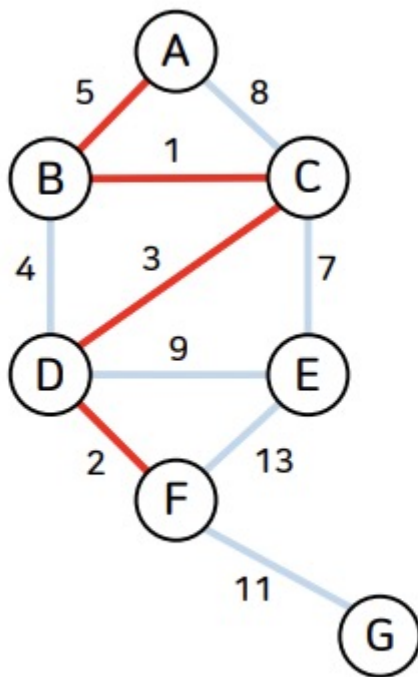


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

A, B는 현재 다른 집합이므로
이 간선을 MST에 포함한다.

가중치 합
6 + 5

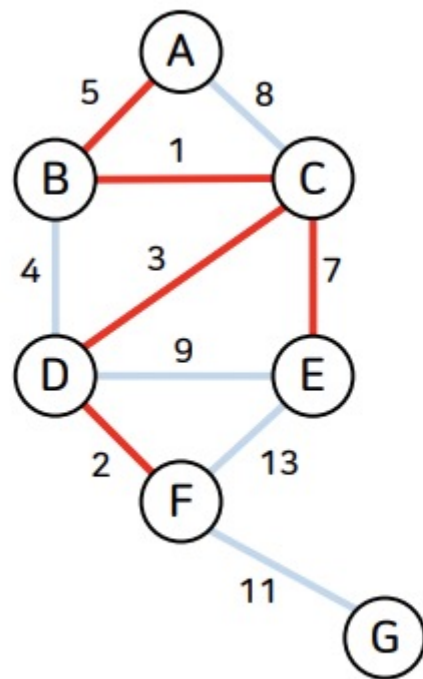


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

C, E는 현재 다른 집합이므로
이 간선을 MST에 포함한다.

가중치 합
11 + 7

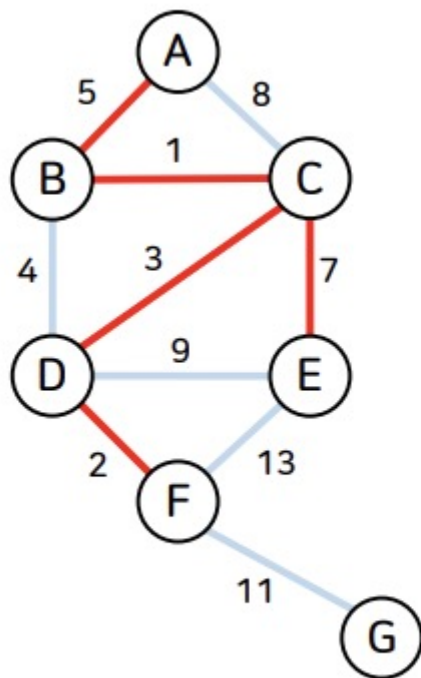


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

A, C는 현재 같은 집합이므로
이 간선은 포함하지 않는다.

가중치 합
18

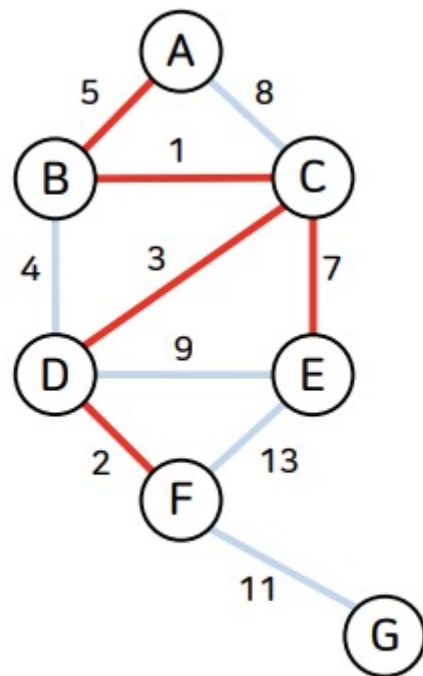


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

C, E는 현재 같은 집합이므로
이 간선은 포함하지 않는다.

가중치 합
18

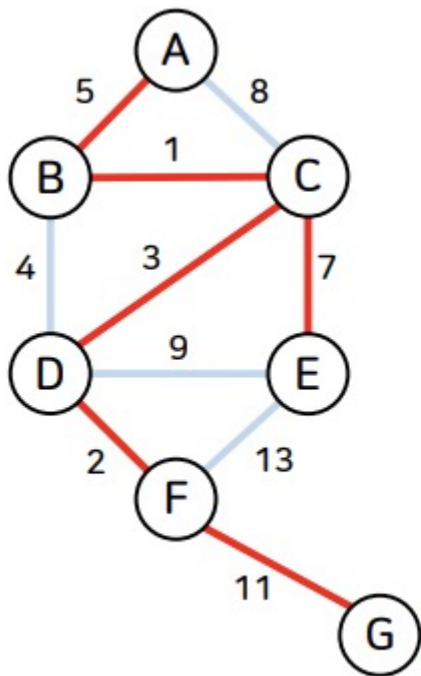


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

F, G는 현재 다른 집합이므로
이 간선을 MST에 포함한다.

가중치 합
18 + 11

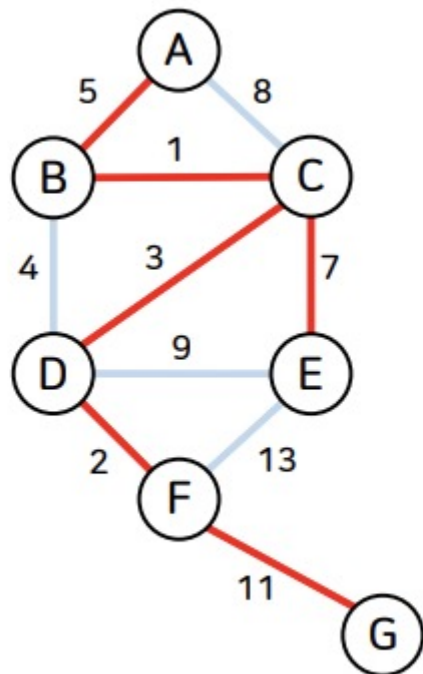


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

E, F는 현재 같은 집합이므로
이 간선은 포함하지 않는다.

가중치 합
29



연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

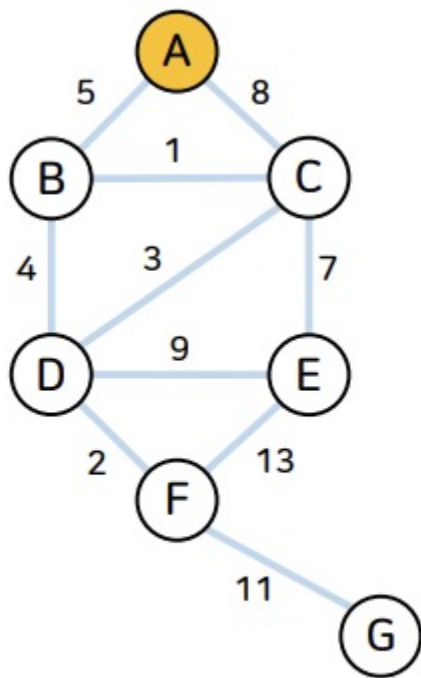
Prim 알고리즘

- 주어진 그래프에서 MST를 만드는 알고리즘
- 단계는 다음과 같다.
 1. 임의의 점 하나를 정하고 그 정점을 MST에 포함시킨다.
 2. 현재 MST에 포함되지 않는 정점과, 포함된 정점을 연결하는 간선 중에서 가중치가 가장 작은 간선을 골라 MST에 포함시킨다.
 3. 2를 반복한다.

어떻게 동작할까?

시작점은 A로 한다.

가중치 합
0

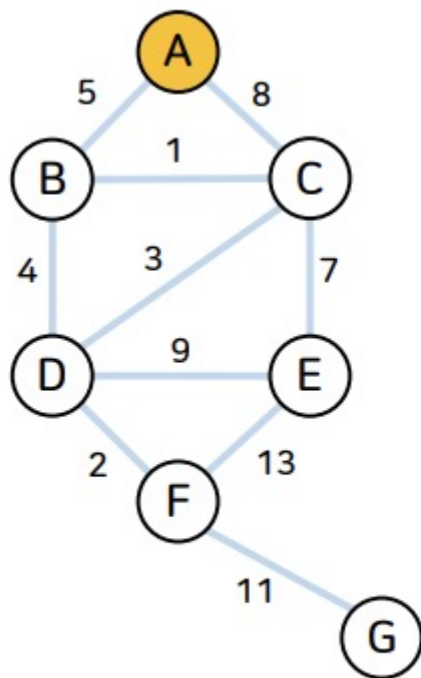


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

A에서 출발하는 모든 간선을 집합에 포함한다.
편의상, 표에서 푸르게 표시된 간선을
집합에 포함된 간선으로 취급한다.

가중치 합
0

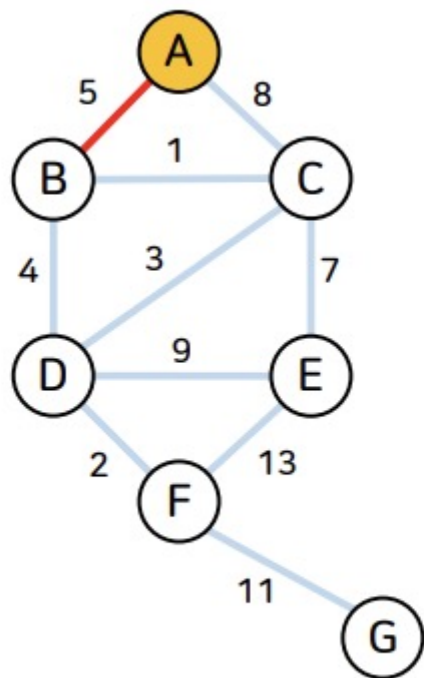


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

현재 집합에 포함된 간선 중,
가장 가중치가 작은 간선은 (A, B)이다.
이를 MST에 포함한다.

가중치 합
0 + 5

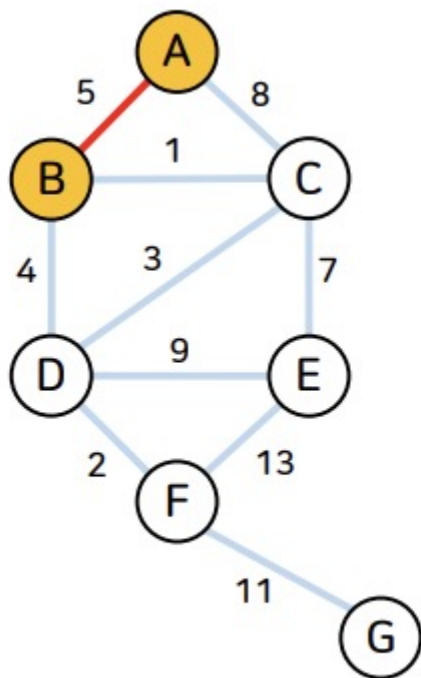


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

B는 MST에 포함되었다.
이제 B에서 출발하는 모든 간선을 집합에 넣는다.

가중치 합
5

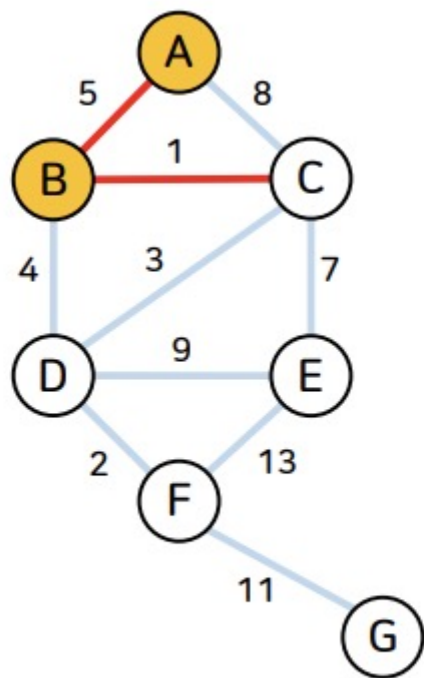


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

현재 집합에 포함된 간선 중,
가장 가중치가 작은 간선은 (B, C)이다.
이를 MST에 포함한다.

가중치 합
5 + 1

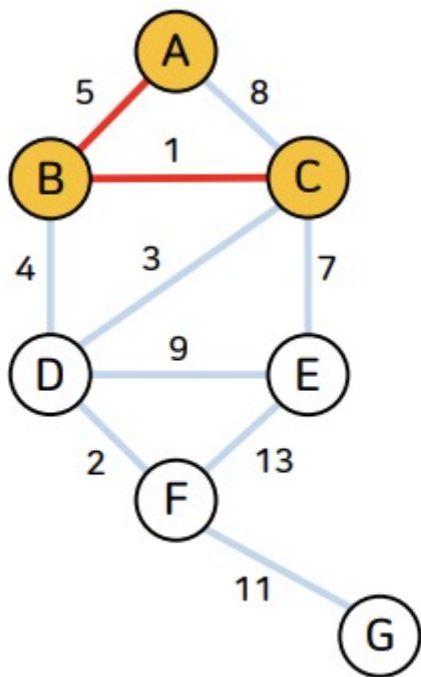


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

C는 MST에 포함되었다.
이제 C에서 출발하는 모든 간선을 집합에 넣는다.

가중치 합
6

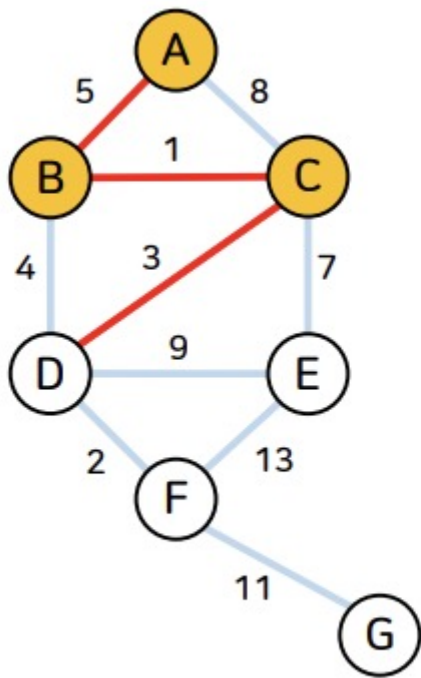


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

현재 집합에 포함된 간선 중,
가장 가중치가 작은 간선은 (C, D)이다.
이를 MST에 포함한다.

가중치 합
6 + 3

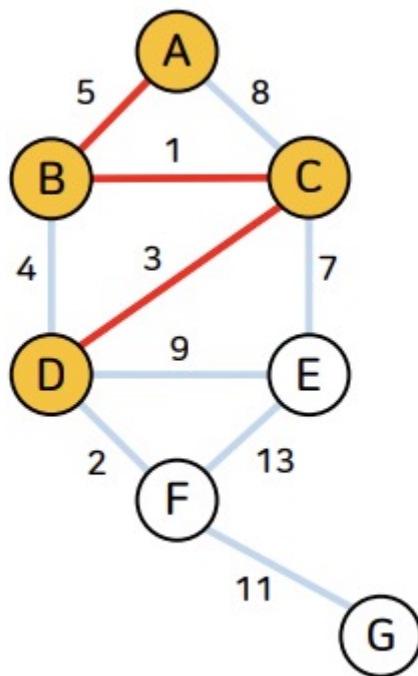


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

D는 MST에 포함되었다.
이제 D에서 출발하는 모든 간선을 집합에 넣는다.

가중치 합
9

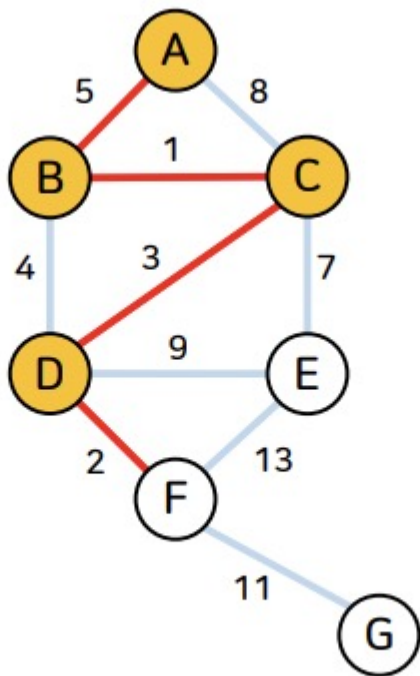


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

현재 집합에 포함된 간선 중,
가장 가중치가 작은 간선은 (D, F)이다.
이를 MST에 포함한다.

가중치 합
9 + 2

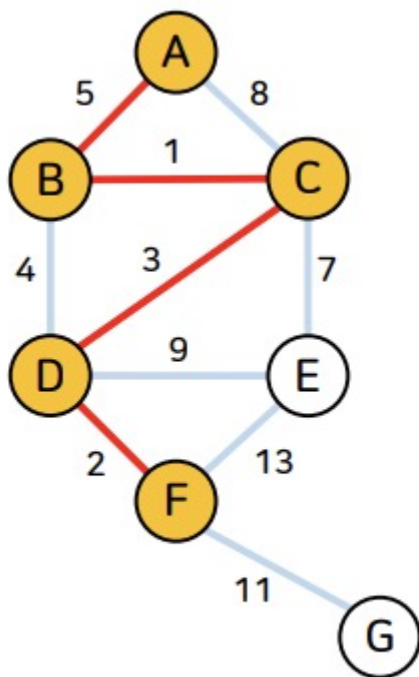


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

F는 MST에 포함되었다.
이제 F에서 출발하는 모든 간선을 집합에 넣는다.

가중치 합
11

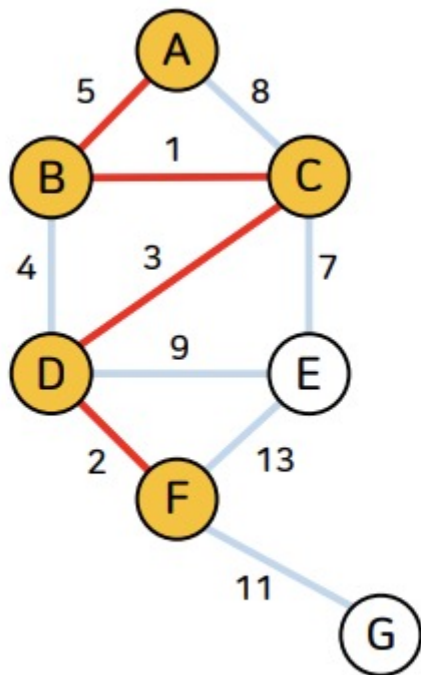


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

현재 집합에 포함된 간선 중,
가장 가중치가 작은 간선은 (B, D)이다.
하지만, B와 D 모두
이미 MST에 포함되었으므로 무시한다.

가중치 합
11

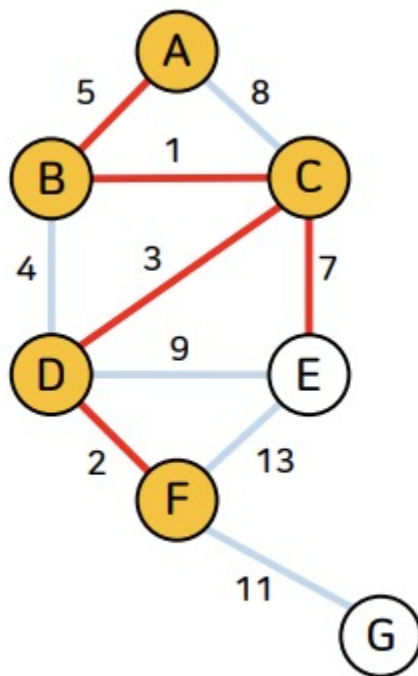


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

현재 집합에 포함된 간선 중,
가장 가중치가 작은 간선은 (C, E)이다.
이를 MST에 포함한다.

가중치 합
11 + 7

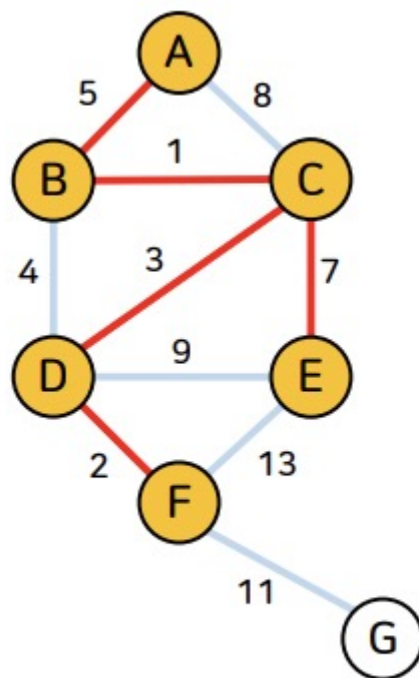


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

E는 MST에 포함되었다.
이제 E에서 출발하는 모든 간선을 집합에 넣는다.

가중치 합
18

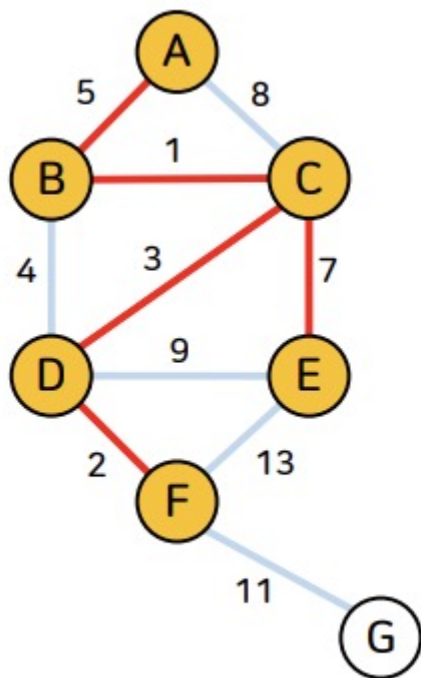


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

현재 집합에 포함된 간선 중,
가장 가중치가 작은 간선은 (A, C)이다.
하지만, A와 C 모두
이미 MST에 포함되었으므로 무시한다.

가중치 합
18

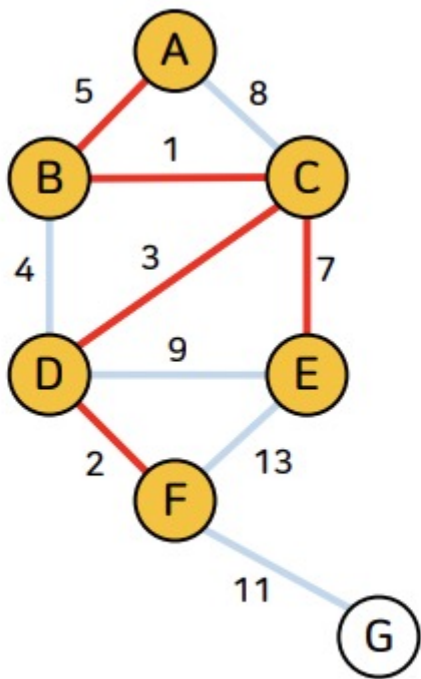


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

현재 집합에 포함된 간선 중,
가장 가중치가 작은 간선은 (D, E)이다.
하지만, D와 E 모두
이미 MST에 포함되었으므로 무시한다.

가중치 합
18

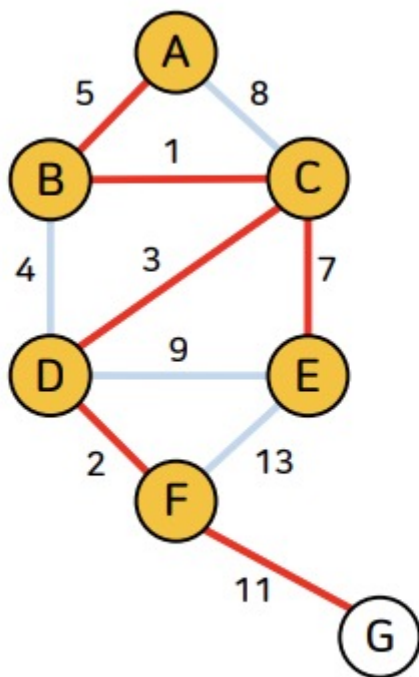


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

현재 집합에 포함된 간선 중,
가장 가중치가 작은 간선은 (F, G)이다.
이를 MST에 포함한다.

가중치 합
18 + 11

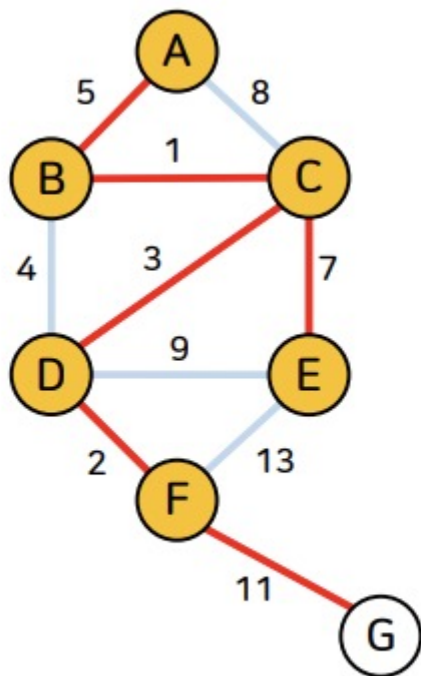


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

현재 집합에 포함된 간선 중,
가장 가중치가 작은 간선은 (E, F)이다.
하지만, E와 F 모두
이미 MST에 포함되었으므로 무시한다.

가중치 합
29



연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13



Kruskal vs Prim

둘 다 MST를 구하는 알고리즘이다.

하지만 구현 방식과 전개되는 논리가 다르다.

시간 복잡도의 차이가 있지 않을까?

Kruskal 의 시간 복잡도

- 간선 정렬
 $O(E \log E) = O(E \log(V^2)) = O(E \log V)$
- 모든 간선에 대해 Find 와 Union
 $E \times O(\log V) = O(E \log V)$

총 시간 복잡도는 $O(E \log V)$ 이다.

Prim의 시간 복잡도

- 최대 E번의 Priority Queue 간선 정보 pop
 $E \times O(\log E) = E \times O(\log V) = O(E \log V)$
- 인접리스트를 통해 간선 정보를 PQ에 push
 $E \times O(\log E) = E \times O(\log V) = O(E \log V)$

총 시간복잡도는 $O(E \log V)$ 가 된다.



성능이 완전히 같다고는 못하지만,
대략적으로는 비슷하다는 것을 알 수 있다.

풀어볼까유



#1197

최소 스패닝 트리

MST 연습문제

풀어볼까유



#1647

도시 분할 계획

MST 연습문제

풀어볼까유



#14621

나만 안되는 연애

MST 연습문제

풀어볼까유



#1045
도로

MST 연습문제

#1197 최소 스패닝 트리

입력 받은 간선 정보를 바탕으로

Kruskal 또는 Prim 알고리즘을 통해 가중치 합을 구해주면 된다.

#1647 도시 분할 계획

MST라는 것을 생각하기 어렵지만, 우리는 트리의 성질을 생각해야한다.
트리는 어느 간선이든 하나 지우면 두개의 트리로 나누어진다.

우리는 두 개의 트리를 두 개의 마을로 생각할 수 있다.

MST를 구하면서 MST의 가중치 합에서 M을 빼주면 된다.

(M은 MST를 구성하는 간선 가중치 중 가장 큰 값)

#14621 나만 안되는 연애

MST 알고리즘을 구현하되, 간선을 걸러서 저장해야한다.

간선을 입력 받으면서 간선이 연결하는 대학이 남초/여초가 맞는지 확인한다.

MST를 구한 뒤, 모든 정점이 같은 집합에 있는지 확인해야한다.

#1045 도로

MST 문제이다. 우리는 문제에 명시된 우선순위를 가지고 간선을 정렬한다.

모든 도시가 연결되어야 하므로 MST를 구해야 한다.

간선을 M 개 포함해야 하기때문에 MST를 구하고 남은 간선 중 우선순위가 높은 $M - (N - 1)$ 개의 간선 가중치를 더해준다.

마지막으로 모든 도시가 같은 집합에 속하는지 확인 해줘야 한다.

풀어볼까유



#17490

일감호에 다리 놓기

MST 응용문제

풀어볼까유



#13344

Chess Tournament

MST 응용문제

풀어볼까유



#17132

두더지가 정보섬에
올라온 이유

MST 응용문제

풀어볼까유



#6416
트리인가?

MST 응용문제



다음 시간에 만나요~