05주차: Disjoint Set & MST

강사: 박정호

챕터 0: 그래프와 트리

그래프와 트리라고 하는 자료구조에 대해 되짚어보자!

그래프란?

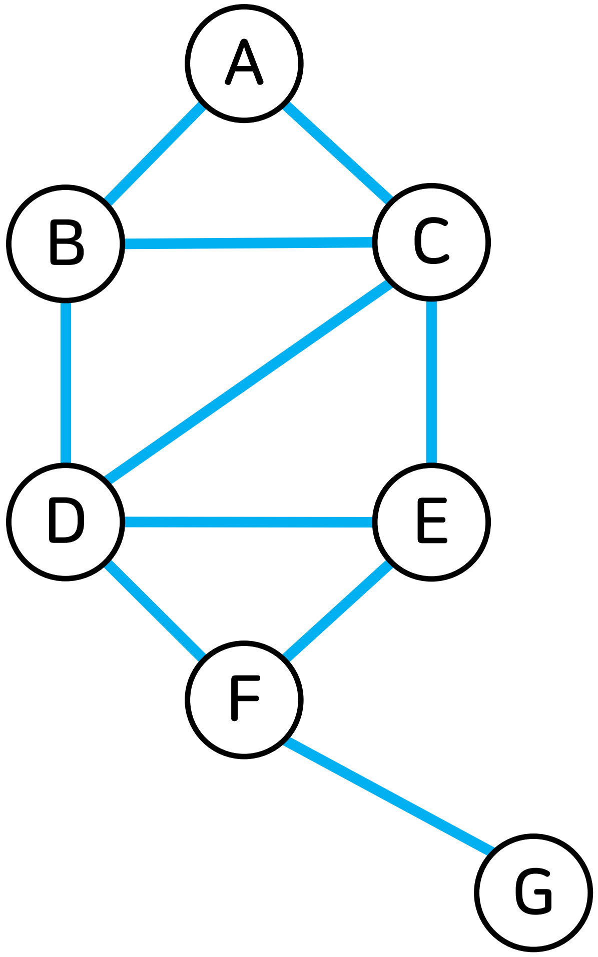
- 일부 객체들의 쌍들이 서로 연관된 객체의 집합을 이루는 구조

  >>> 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

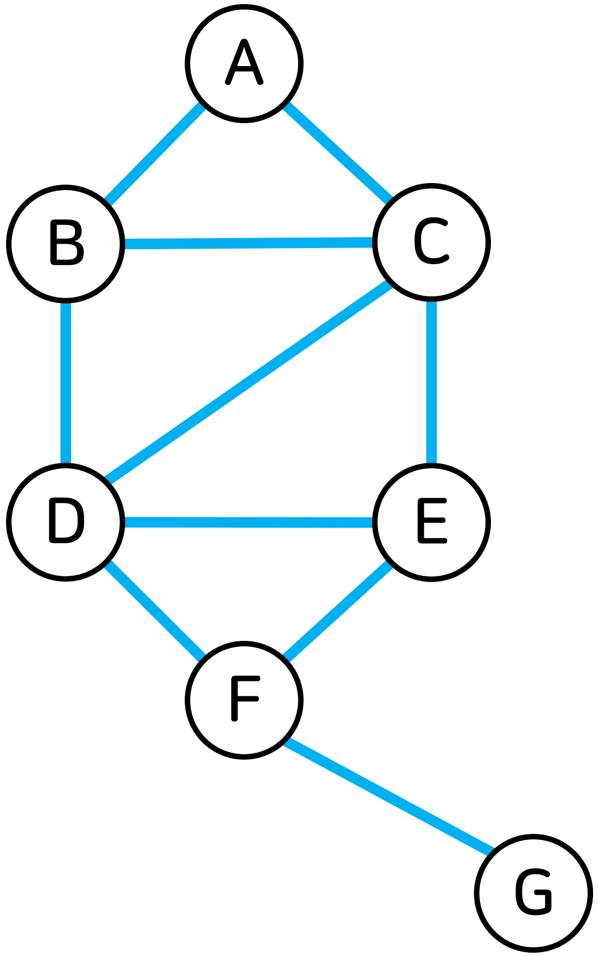
- 그래프가 가지는 특성에 따라
  가중치 그래프, 단방향 그래프, 양방향 그래프 등등이 있다.



가중치가 없는 양방향 그래프

상식으로 알아두는 그래프 용어

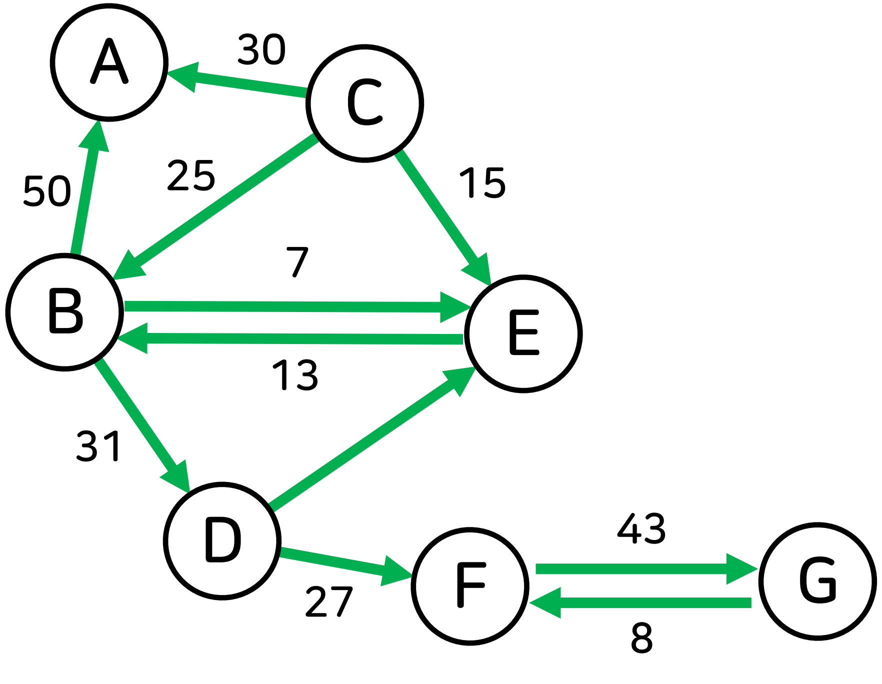
- 정점 == 노드 == vertex
- 간선 == edge
- 가중치 == weight



가중치가 없는 양방향 그래프

가중치란?

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



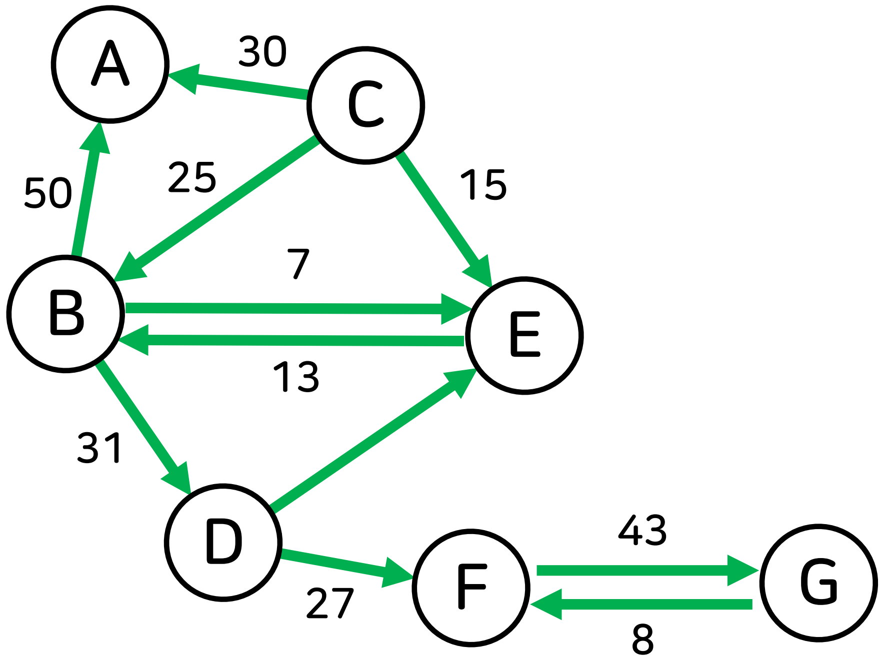
가중치가 있는 단방향 그래프

가중치의 예시(아래의 모든 예시는 옆의 그래프를 참고합니다.)

- 두 지점 간의 거리
  ex) B에서 A로 가는 도로의 길이는 50이다.
  C에서 B로 가는 도로의 길이는 25이다.

- 이동하는 데 드는 비용
  ex) B에서 E로 가는 것은 13만큼의 비용이 든다.
  F에서 G로 가는 것은 43만큼 시간이 든다.

- 여타 여러가지 “연결”에 관한 값들…
  ex) 도로 설치 비용, 이동 시간....

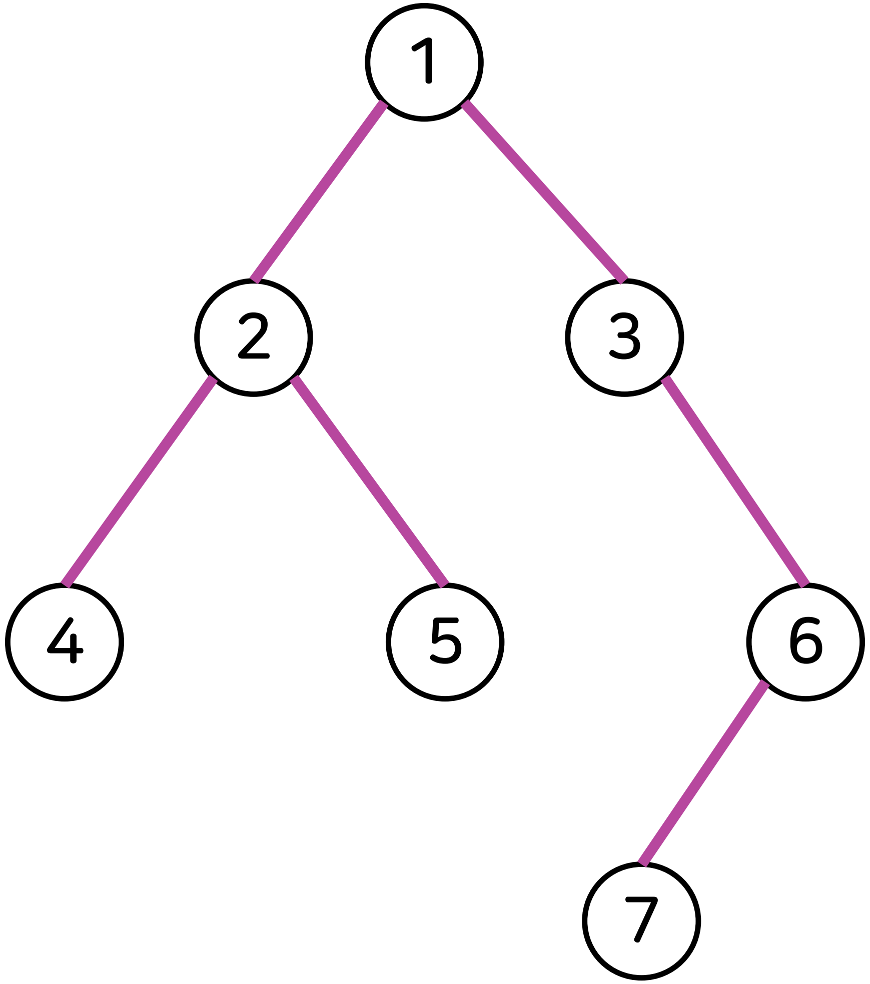


가중치가 있는 단방향 그래프

트리란?

- 사이클이 없는 연결 그래프

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



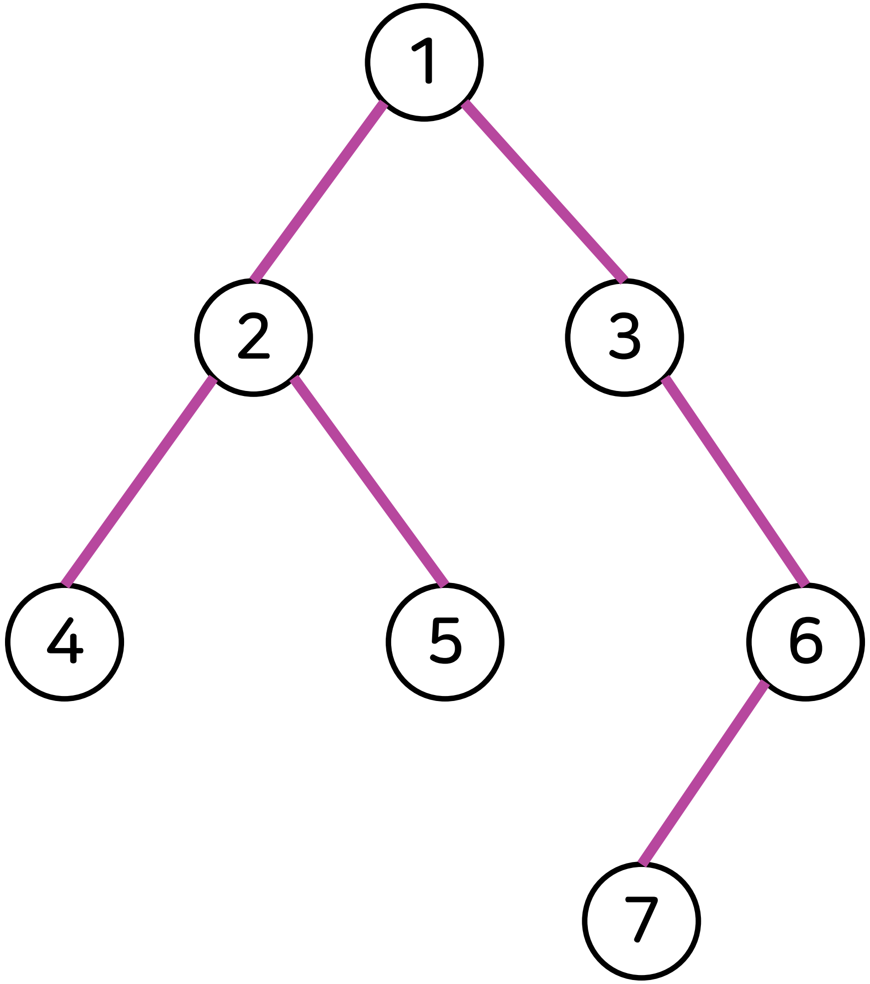
상식으로 알아두는 트리 용어

- 부모 노드와 자식 노드 :
  어떤 노드가 가리키는 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.

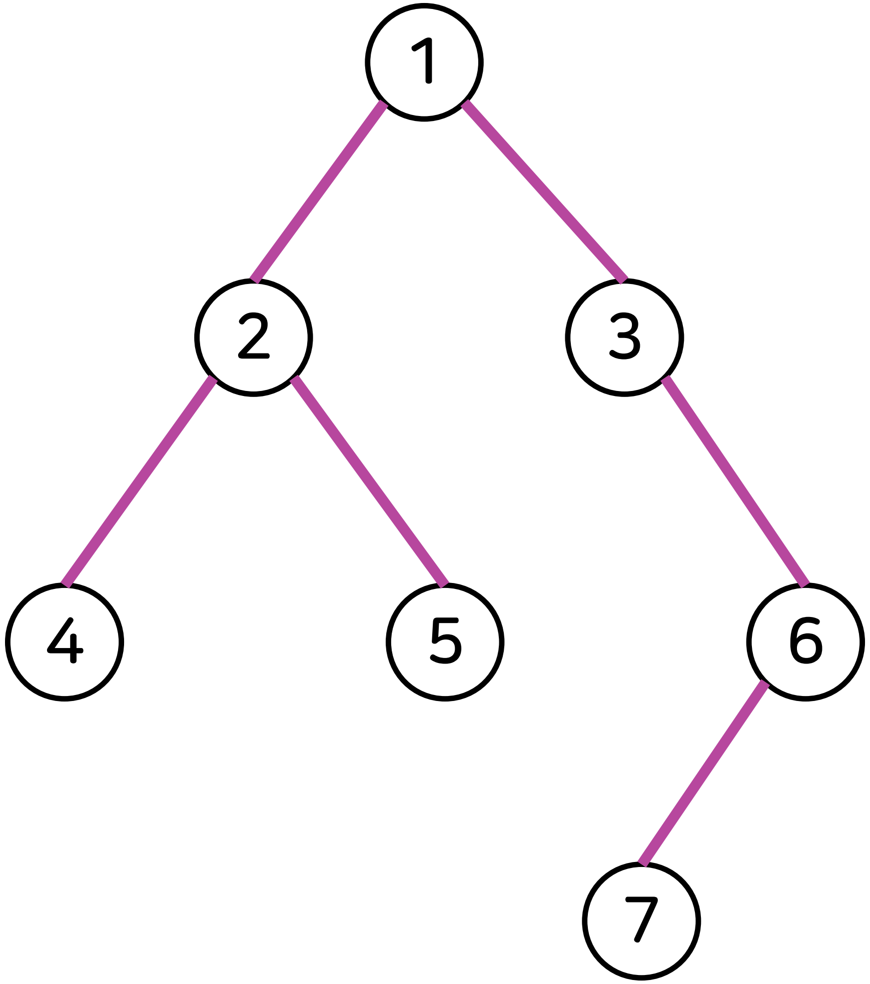
- root 노드 :
  부모 노드가 없는 노드, 최상위 노드
  ex) 옆의 트리의 root 노드는 1이다.

- Leaf 노드 :
  자식이 없는 노드
  ex) 옆의 트리의 leaf 노드는 4, 5, 7이다.



트리의 특징

- 모든 트리는 하나의 root 노드를 갖는다.
- 모든 노드는 root 노드와 연결된 유일한 경로를 갖는다.
- root 노드 이외의 모든 노드는 부모 노드를 갖는다.
- 트리는 (노드 개수) – 1개의 간선 갖는다.
  Root 노드 이외의 모든 노드가 부모 노드를 갖고,
  부모 노드와 자식 노드는 간선으로 이어져 있기 때문이다.

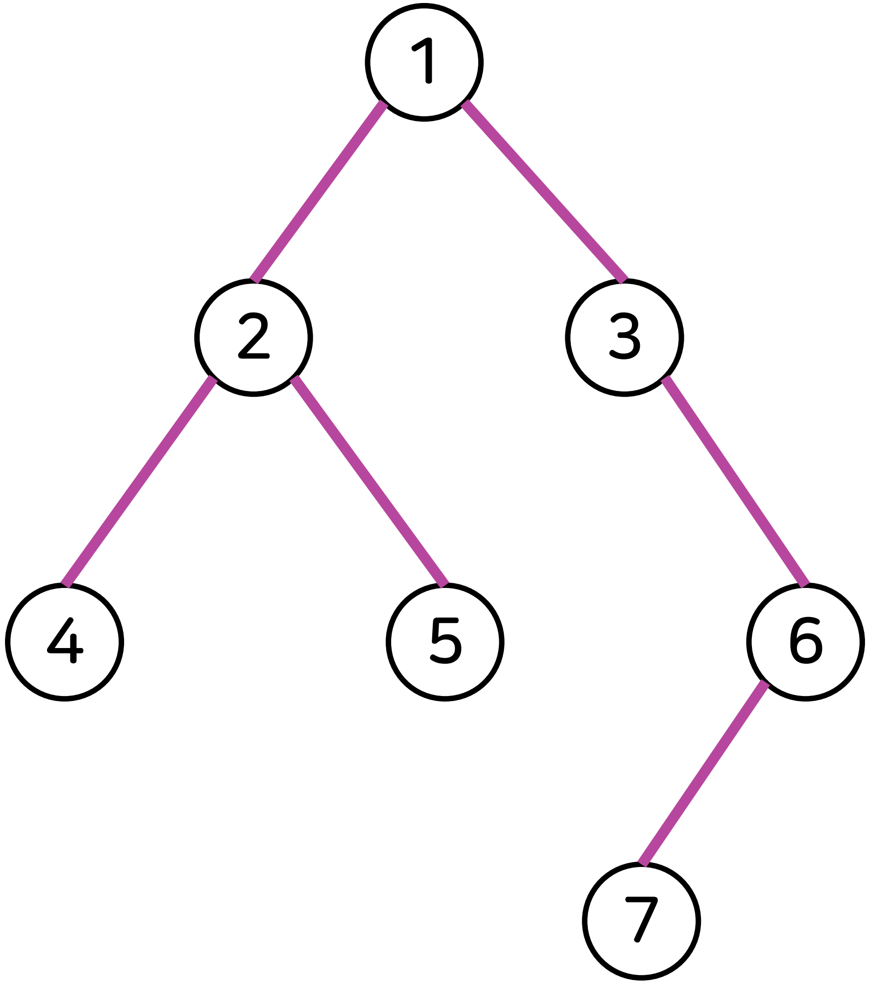


챕터 1: Disjoint Set

분리 집합? Disjoint Set에 대해서 알아보자!

시작하기 전에…

- Disjoint Set 구현을 위해 트리의 특징을 되짚어보자.
  - 모든 트리는 하나의 root 노드를 갖는다.
  - 모든 노드는 root 노드와 연결된 유일한 경로를 갖는다.
  - Root 노드 이외의 모든 노드는 단 하나의 부모 노드를 갖는다.



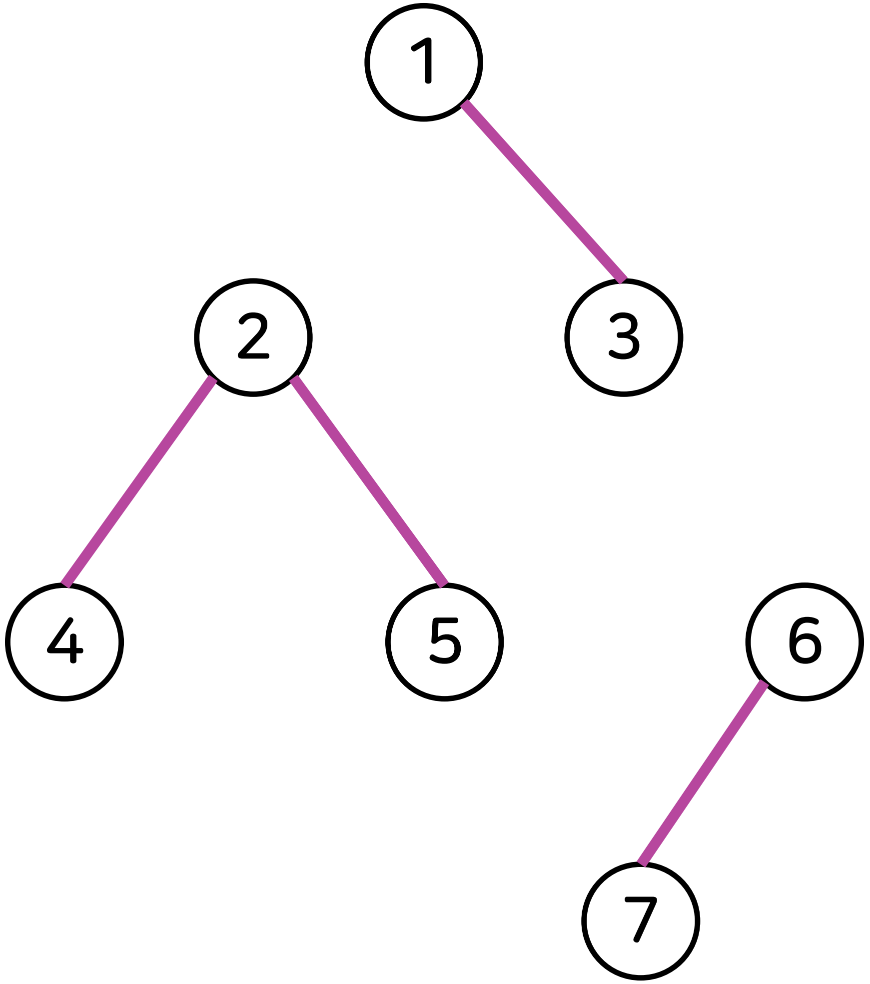
Disjoint Set?

- 서로소 부분 집합을 관리하는 자료구조

  * 서로소 부분 집합? :
  어떤 집합의 부분 집합 중 겹치는 원소가 없는 부분 집합들

  >>> 그룹을 묶고, 그 그룹을 관리하기 위해 사용하는 자료구조

- 기본적으로 두가지 연산을 지원한다.
  - UNION
    주어진 두 개의 집합을 하나로 합친다.
  - FIND
    주어진 원소가 어떤 집합에 속해 있는지 찾는다.
    코드 구현 시에는 집합의 원소 중 그 집합을 대표하는 원소를 정해서
    그 값을 return하는 방식으로 진행한다.



서로소 부분 집합 3개

어떻게 구현할까? - 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.

어떻게 구현할까? - IMPLEMENTATION

- 우선 FIND부터 구현해보자.
  FIND 는 주어진 노드가 어느 집합에 속하는지 찾아야 한다.

- 여기서 우리는 트리의 성질 몇가지를 생각해야 한다.
  - 모든 노드는 root 노드와 연결된 유일한 경로를 갖는다.
  - root 노드 이외의 모든 노드는 단 하나의 부모 노드를 갖는다.

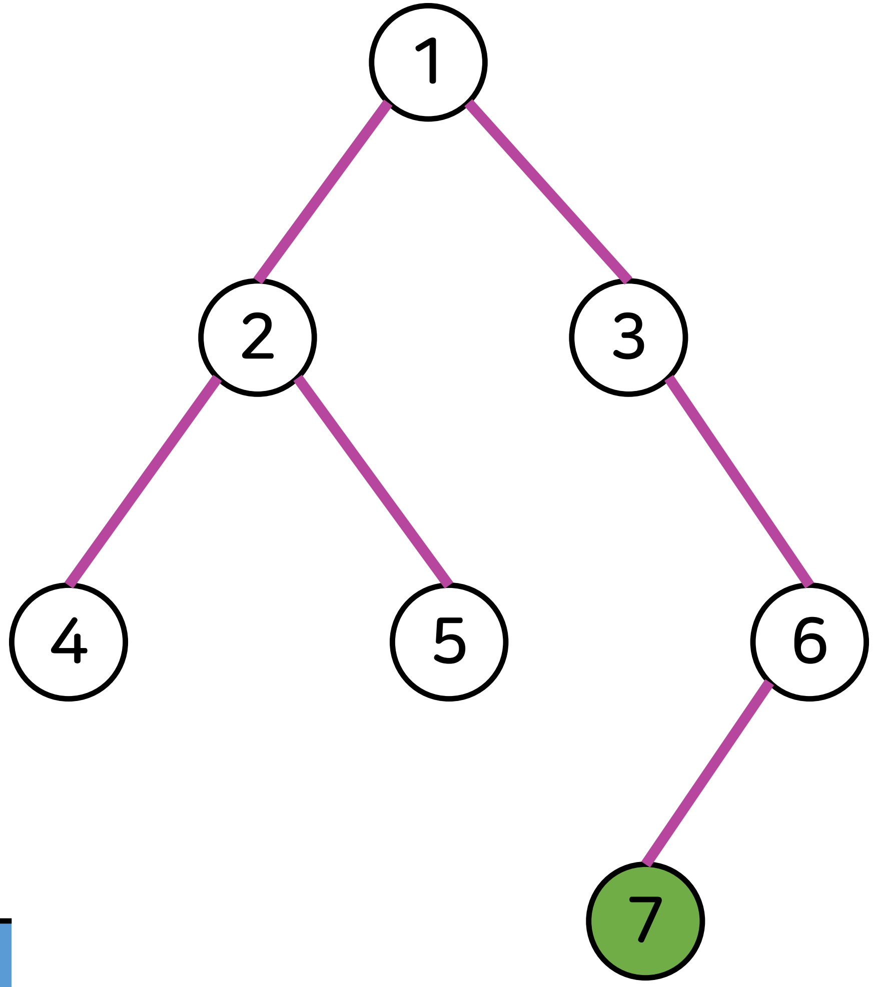
  이 성질을 잘 생각해보면, 주어진 노드의 부모 노드를 찾고,
  그 부모 노드의 부모 노드를 찾고…. 이런 식으로 반복하다 보면
  결국 root까지 도달할 수 있다는 것을 알 수 있다.

어떻게 구현할까? - IMPLEMENTATION

```
int parent[SIZE];//노드의 부모를 저장하는 배열, 처음에 parent[i] == i 이도록 초기화해놓도록 한다.

int FIND(int node) {
		//현재 노드와 현재 노드의 부모 노드가 같다면(즉 내가 root 노드일 경우)
		if (node == parent[node]) {
				//현재 노드를 return
				return node;
		}
		//아니라면
		//내 부모 노드의 부모를 찾아야 한다
		//복잡하게 생각하지 말고, 우리가 수행하는 과정을 그대로 현재 노드의 부모에 적용하면 된다.
		return FIND(parent[node]);
}
```

어떻게 동작할까? - FIND

현재 노드
7

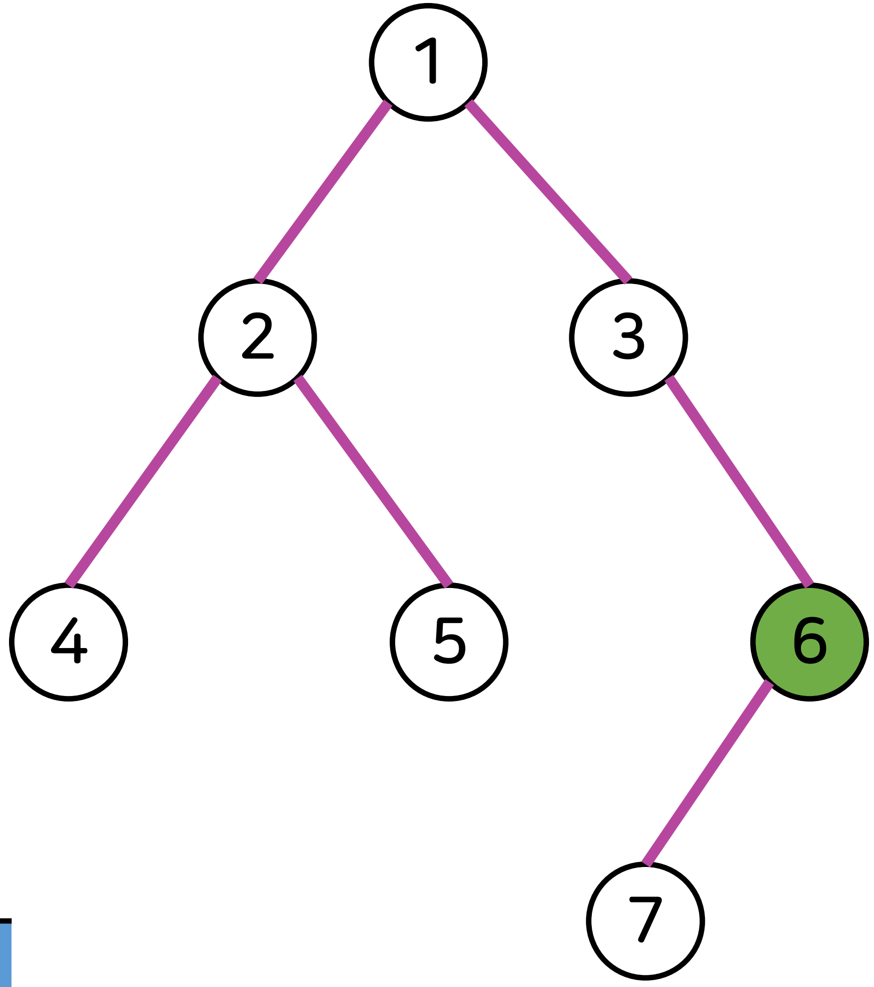


노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	3	6

어떻게 동작할까? - FIND

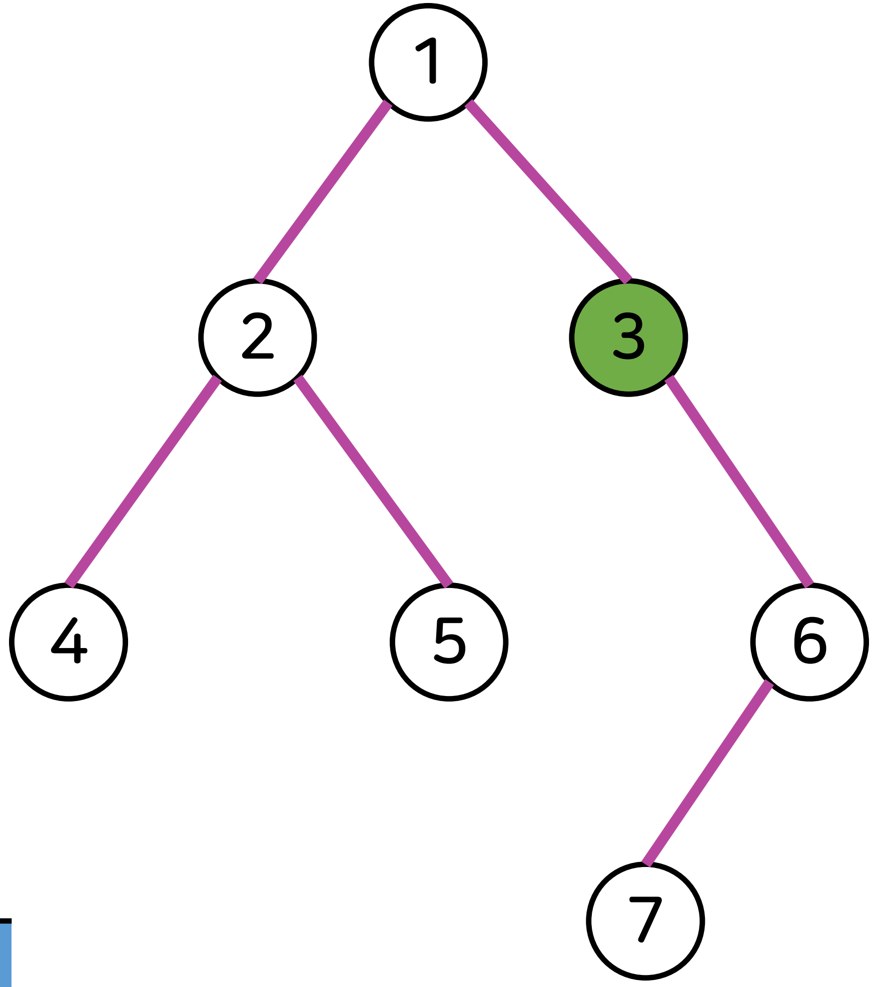
현재 노드
6

노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	3	6



어떻게 동작할까? - FIND

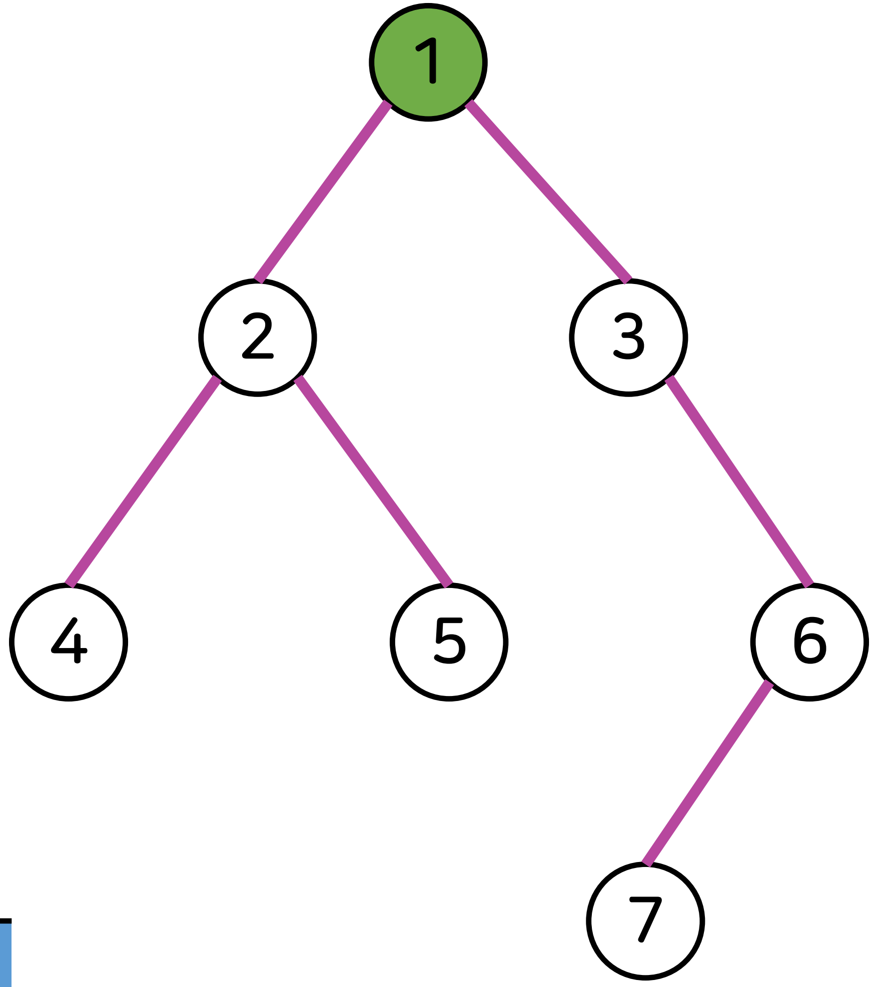
현재 노드
3



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

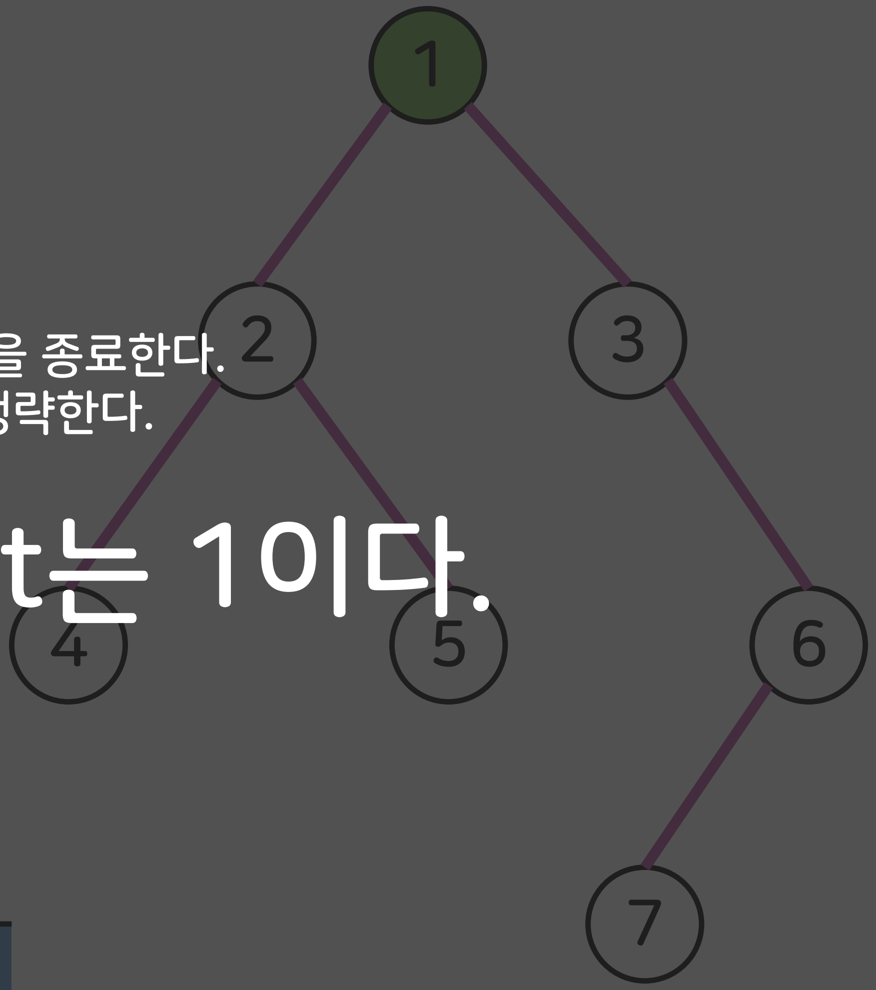
어떻게 동작할까? - FIND

현재 노드와 부모 노드가 같으므로, 탐색을 종료한다.
재귀 호출 후 다시 돌아오는 과정은 생략한다.

따라서, 이 트리의 root는 1이다.

현재 노드
1

노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	3	6



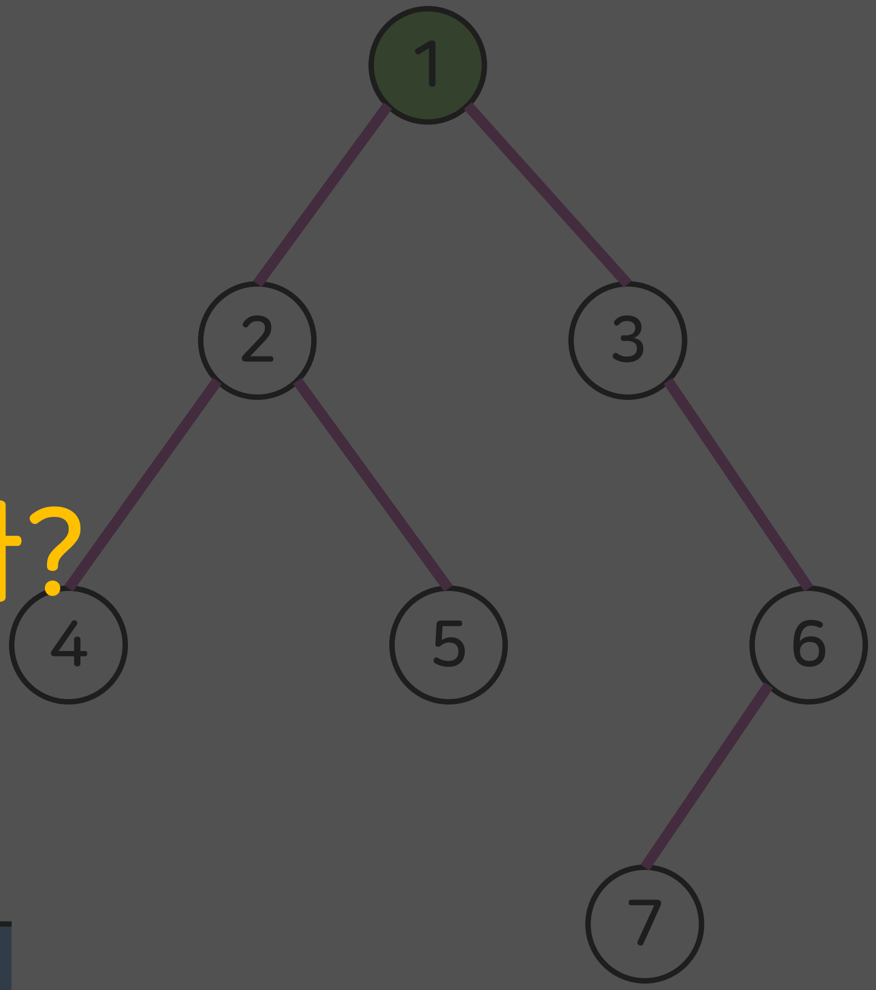
어떻게 동작할까? - FIND

그런데,

이게 최선일까?

현재 노드
1

노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	3	6



어떻게 구현할까? - 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 m인 노드를 가지고 FIND를 n번 사용한다면 어떻게 될까?
  당연히 시간복잡도는 O(n*m)이 될 것이다.
  n = 100000, m = 1000 정도만 되어도 n*m = 1억이므로 너무 오래 걸린다.

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

어떻게 구현할까? - 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.

  <u>“같은 집합에 있다”</u>라는 조건만 유지한다면, 트리의 구조는 얼마든지 변형해도 무방하다.

- 여기까지 이해했다면 어느 정도 답이 보일 것이다.

  root 이외의 모든 노드가 root를 부모로 가질 수 있도록 하면 어떨까?

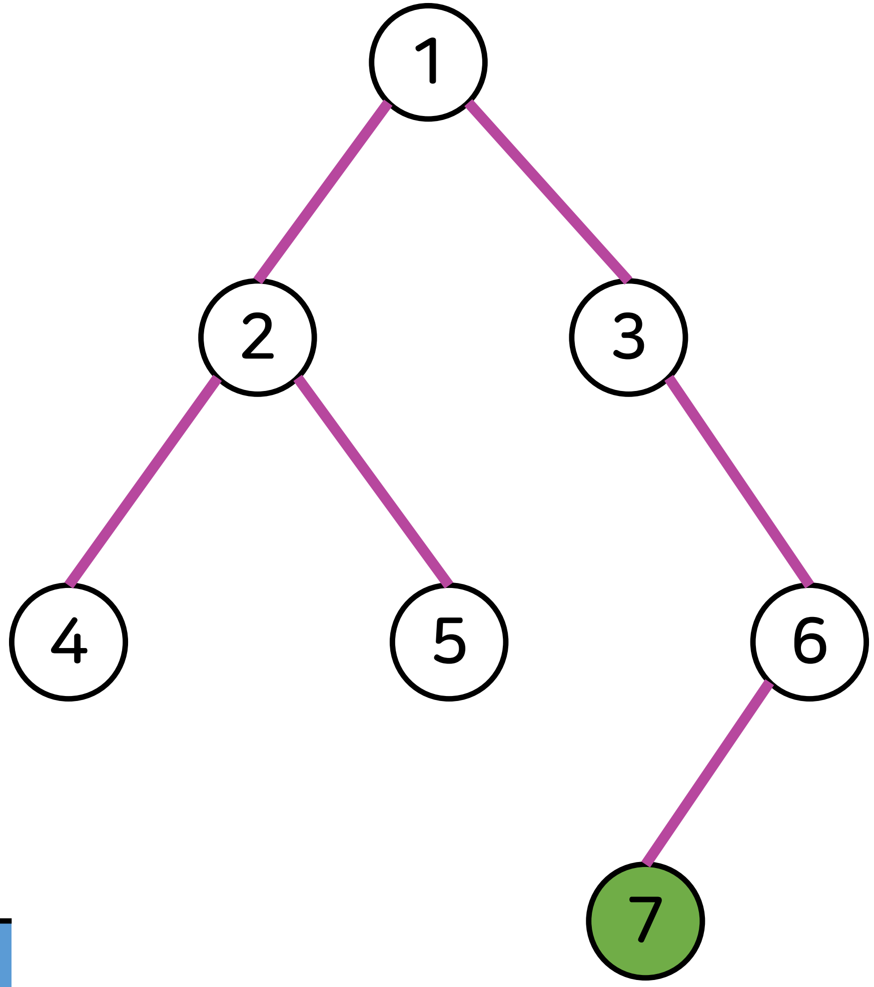
어떻게 구현할까? - IMPLEMENTATION

```
int parent[SIZE];//노드의 부모를 저장하는 배열, 처음에 parent[i] == i 이도록 초기화해놓도록 한다.


int FIND(int node) {
        //현재 노드와 현재 노드의 부모 노드가 같다면(즉 내가 root 노드일 경우)
        if (node == parent[node]) {
                //현재 노드를 return
                return node;
        }
        //아니라면 내 부모 노드의 부모를 찾아야 한다
        //여기서 하나의 트릭이 사용된다.
        //나의 부모를 내 부모의 부모로 바꾸도록 하자. 어차피 모두 같은 집합 소속이므로 문제는 없다.
        return parent[node] = FIND(parent[node]);
}
```

어떻게 동작할까? - FIND

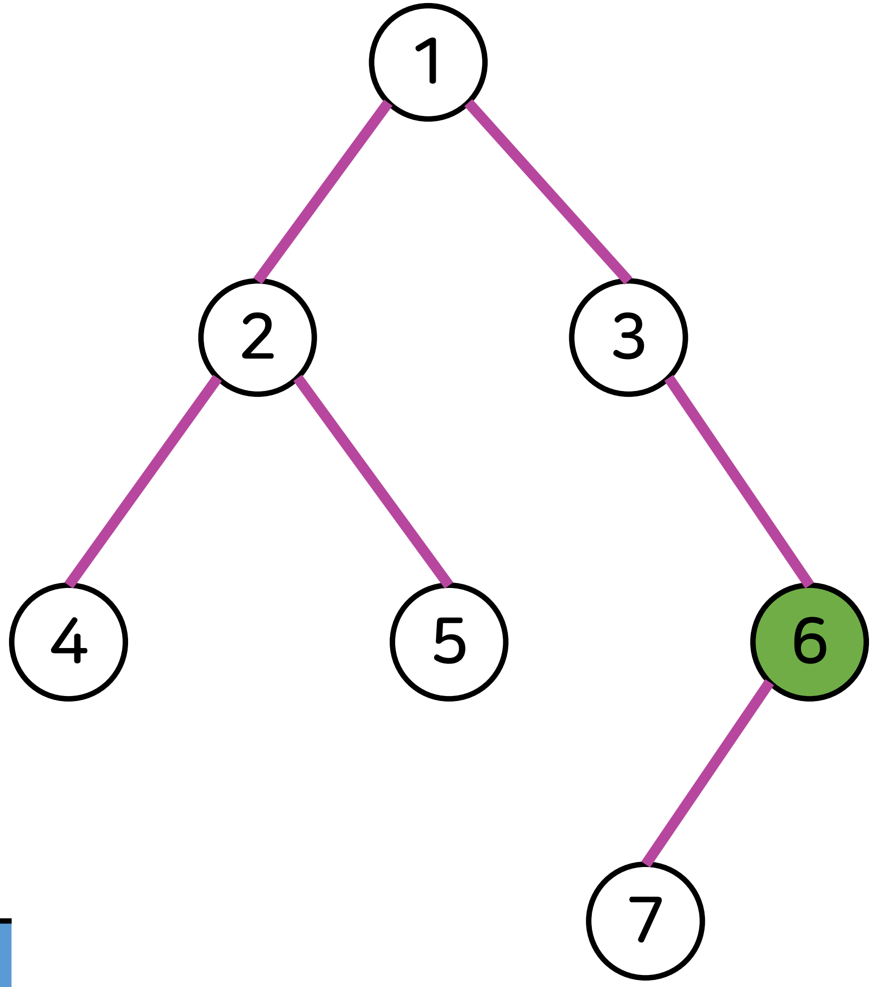
현재 노드
7

노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	3	6



어떻게 동작할까? - FIND

현재 노드
6

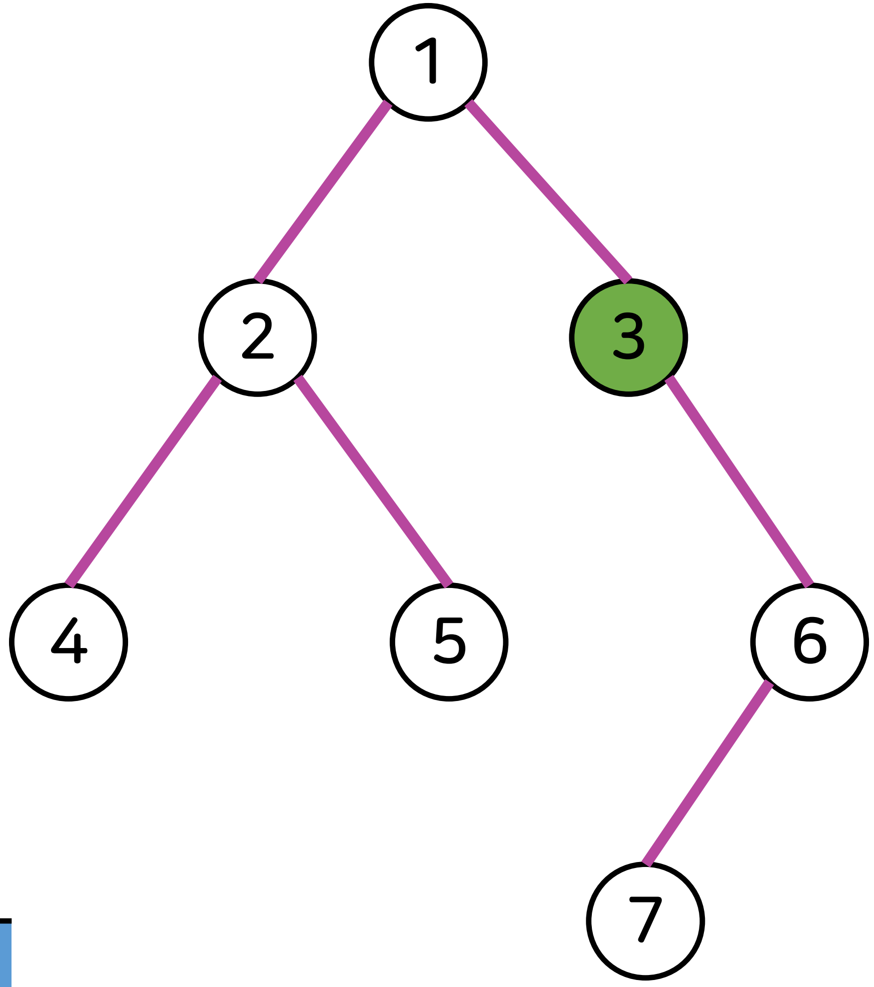


노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	3	6

어떻게 동작할까? - FIND

현재 노드
3

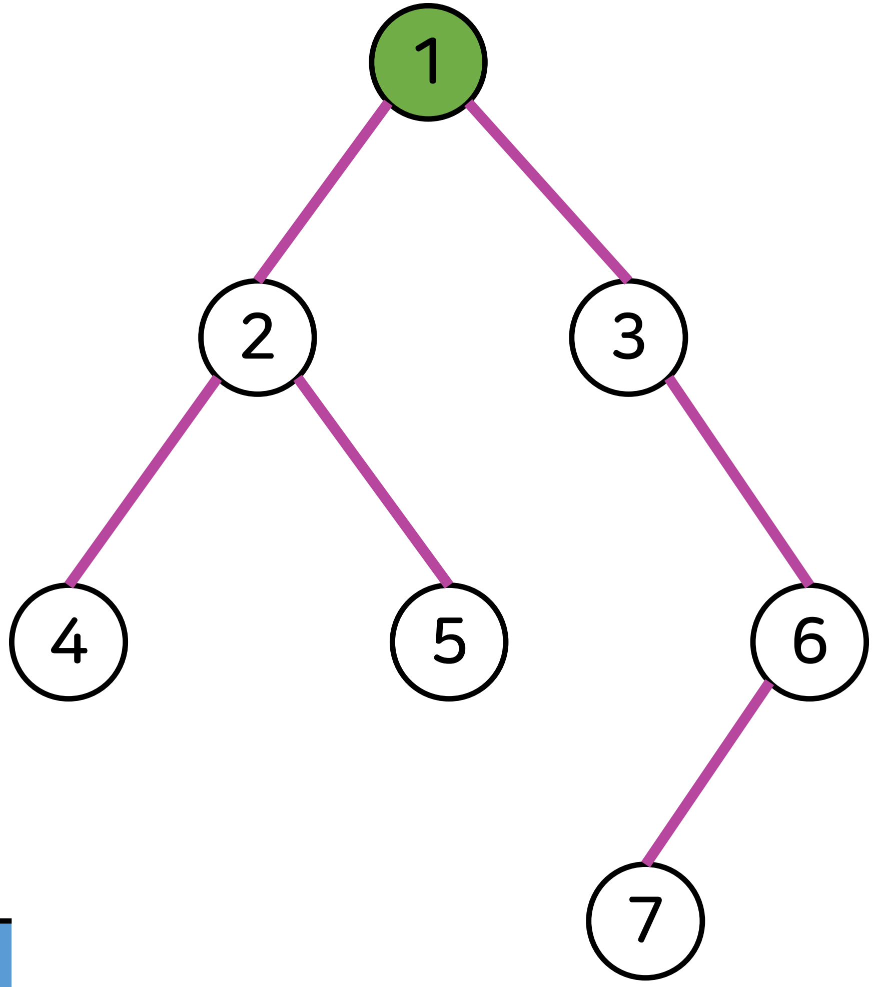
노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	3	6



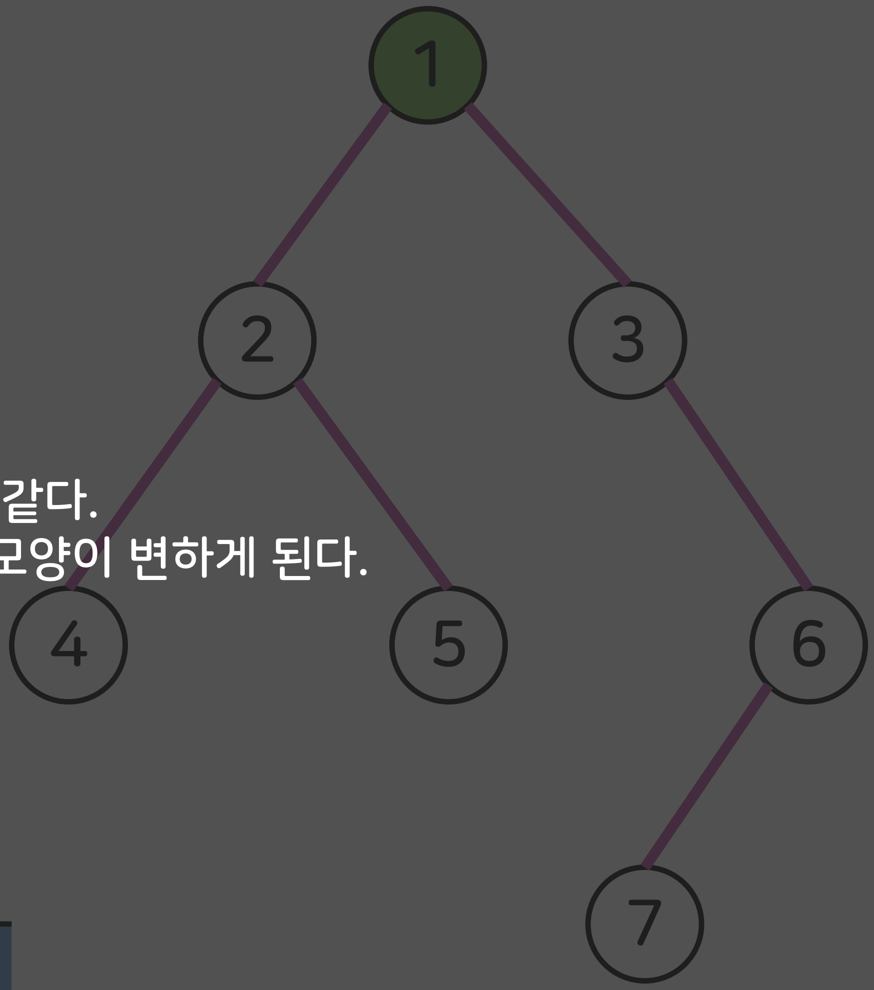
어떻게 동작할까? - FIND

현재 노드
1

노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	3	6



어떻게 동작할까? - FIND



현재 노드
1

여기까지는 앞의 내용과 과정이 같다.
하지만, 재귀 호출이 끝나고 돌아오면서 트리 모양이 변하게 된다.

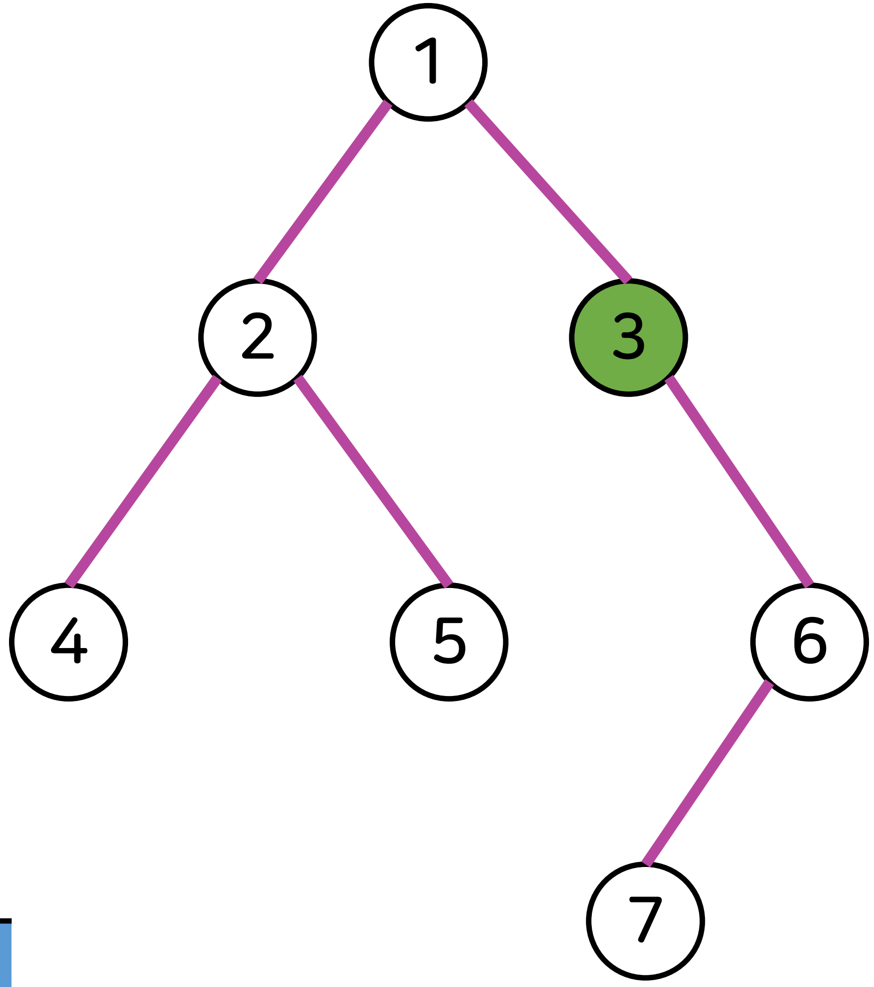
노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	3	6

어떻게 동작할까? - FIND

현재 노드
3

return 값
1

노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	3	6

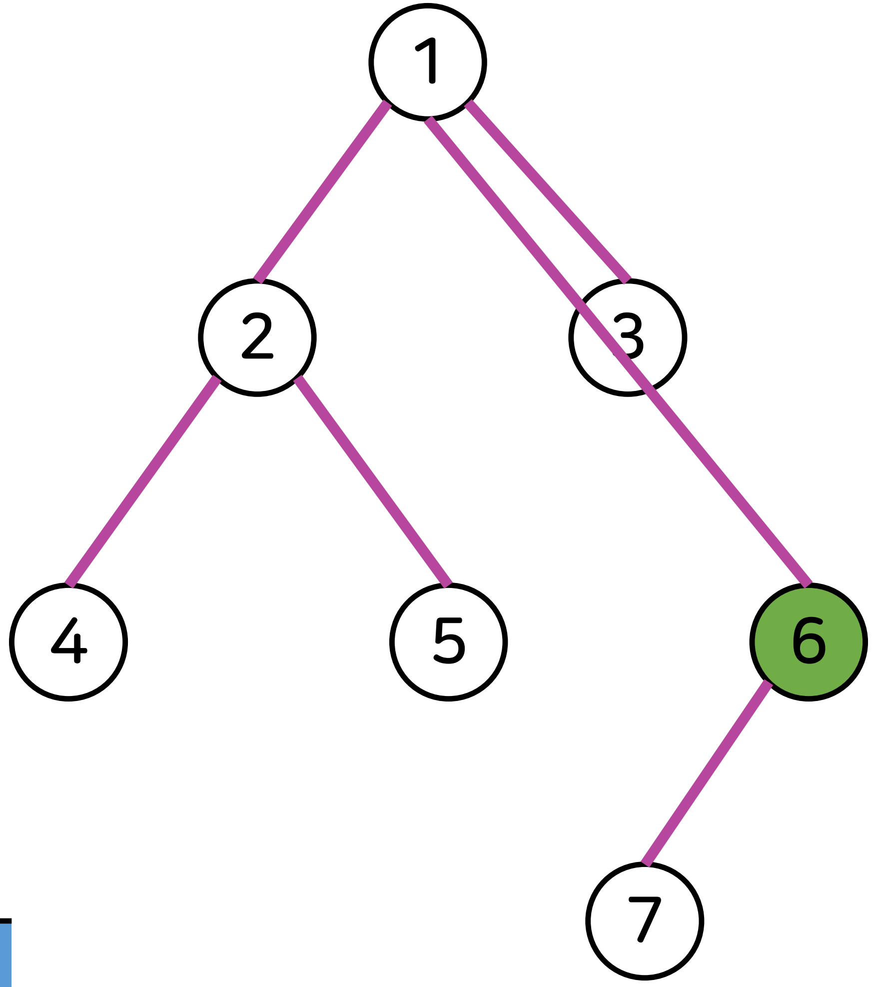


어떻게 동작할까? - FIND

현재 노드
6

return 값
1

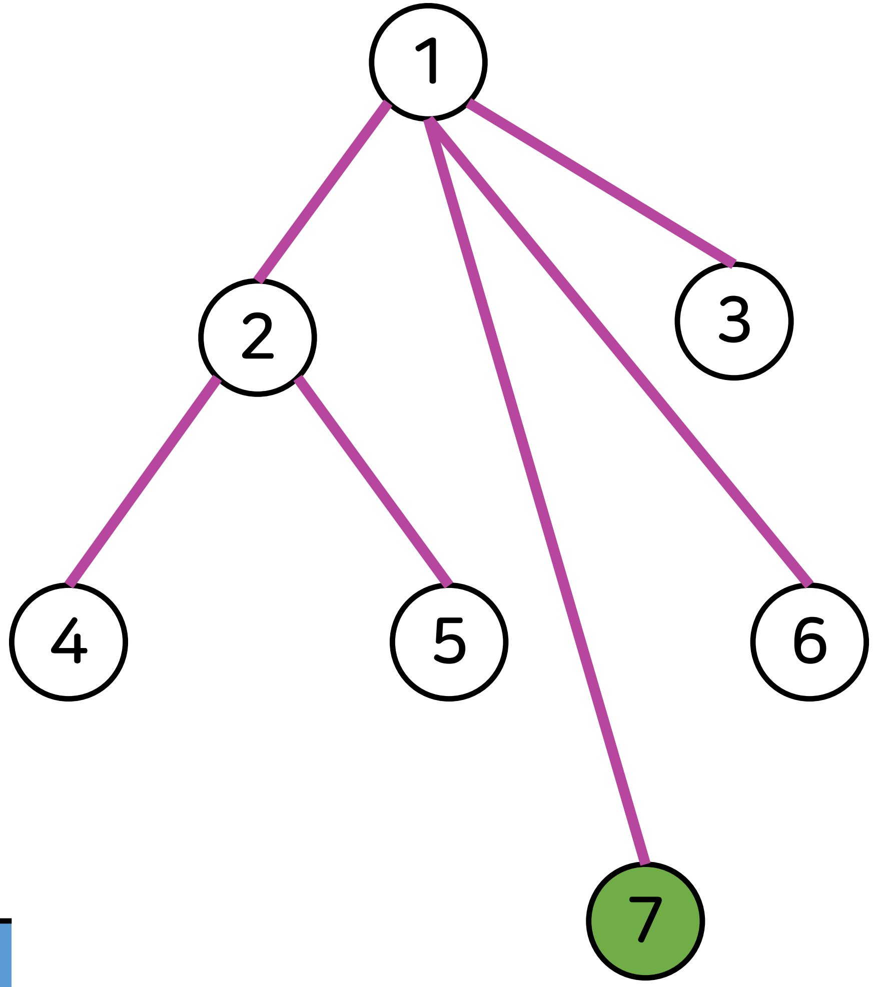
노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	1	6



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



어떻게 동작할까? - FIND

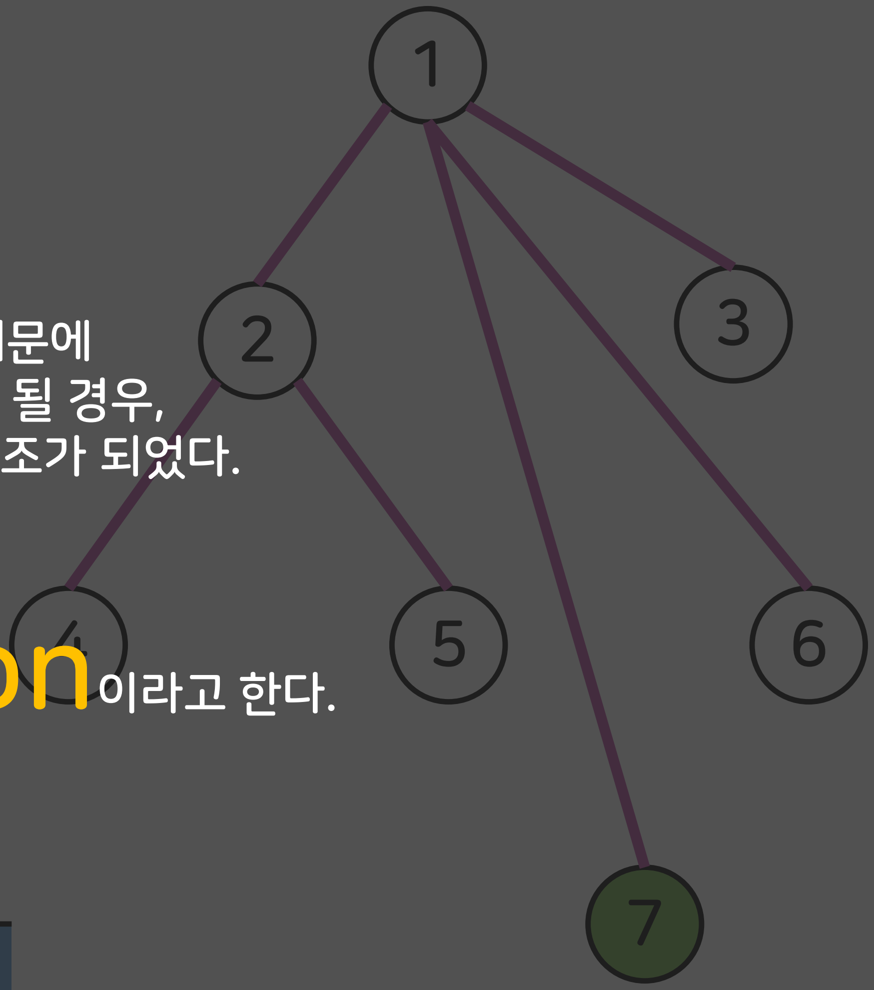
노드 6과 7의 깊이가 1이 되었기 때문에
이후에 FIND(6), FIND(7)을 다시 하게 될 경우,
단 한번의 재귀호출로 답을 찾을 수 있는 구조가 되었다.

현재 노드	return 값
7	1

이러한 방식을

Path Compression이라고 한다.

노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	1	1



어떻게 구현할까? - IMPLEMENTATION

- 이제 UNION을 구현해보자.
  UNION은 두 집합을 하나로 합쳐야 한다.

- 두 집합을 하나로 합치기 위해서는 두 집합의 root를 같게 해야 한다.
  - Root 노드 이외의 모든 노드는 단 하나의 부모 노드를 갖는다.

  두 집합을 전달받았다는 것은 두 트리의 root를 전달받았다고 해석할 수 있다.
  위에서 언급한 트리의 특징을 생각하면 두 root 노드 모두 부모 노드가 없다고 생각할 수 있고,
  두 노드를 a, b라고 하면, a를 b의 자식 노드로 만드는 것으로 두 집합을 합칠 수 있다.

어떻게 구현할까? - IMPLEMENTATION

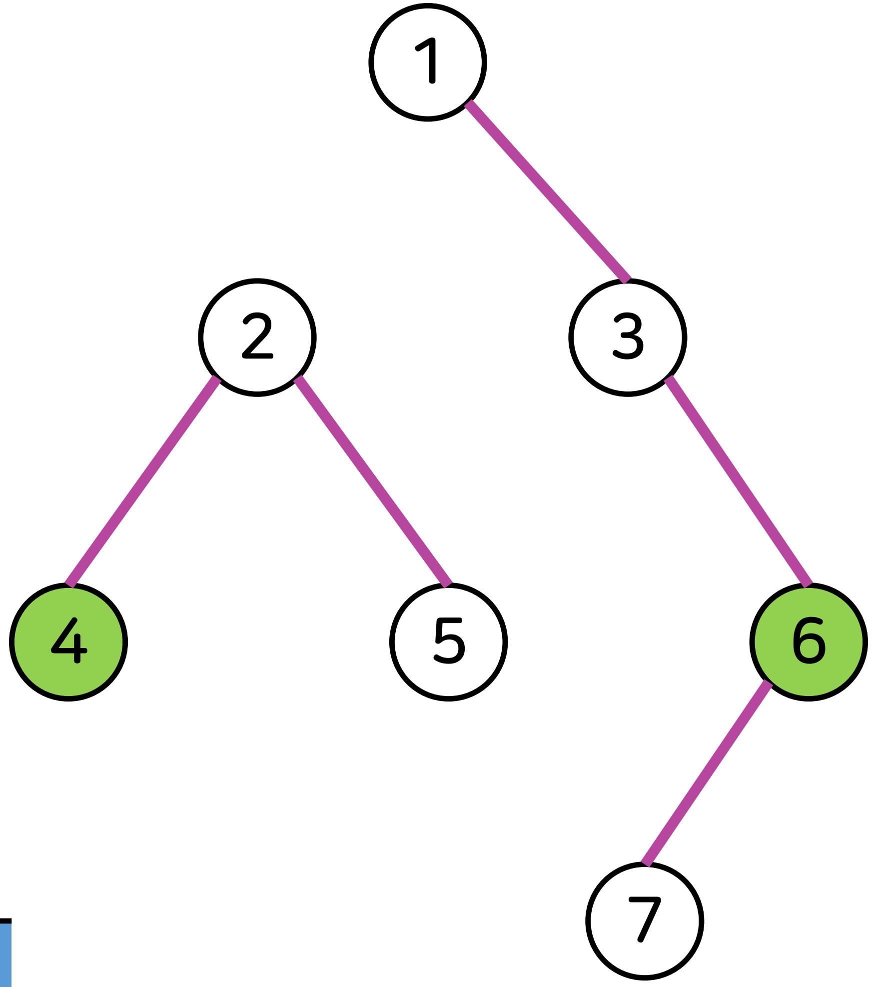
```
void UNION(int node_a, int node_b) {
		//두 노드의 root를 FIND로 구하자
		int root_a = FIND(node_a);
		int root_b = FIND(node_b);

		//root_b의 부모를 root_a로 설정하는 것으로 root_a와 root_b로 대표되는 두 집합을 합친다.
		parent[root_b] = root_a;
}
```

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 a
6

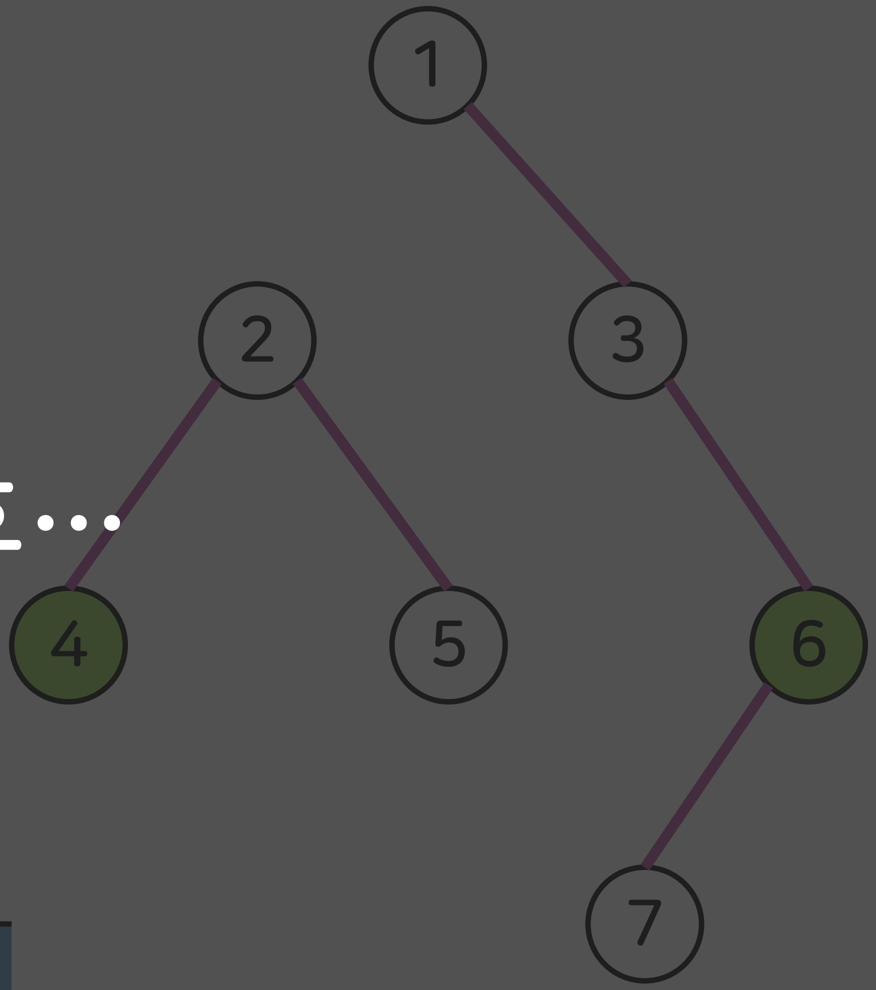
노드 b
4

FIND 연산 이후…

집합 a

집합 b

노드	1	2	3	4	5	6	7
부모	1	2	1	2	2	3	6

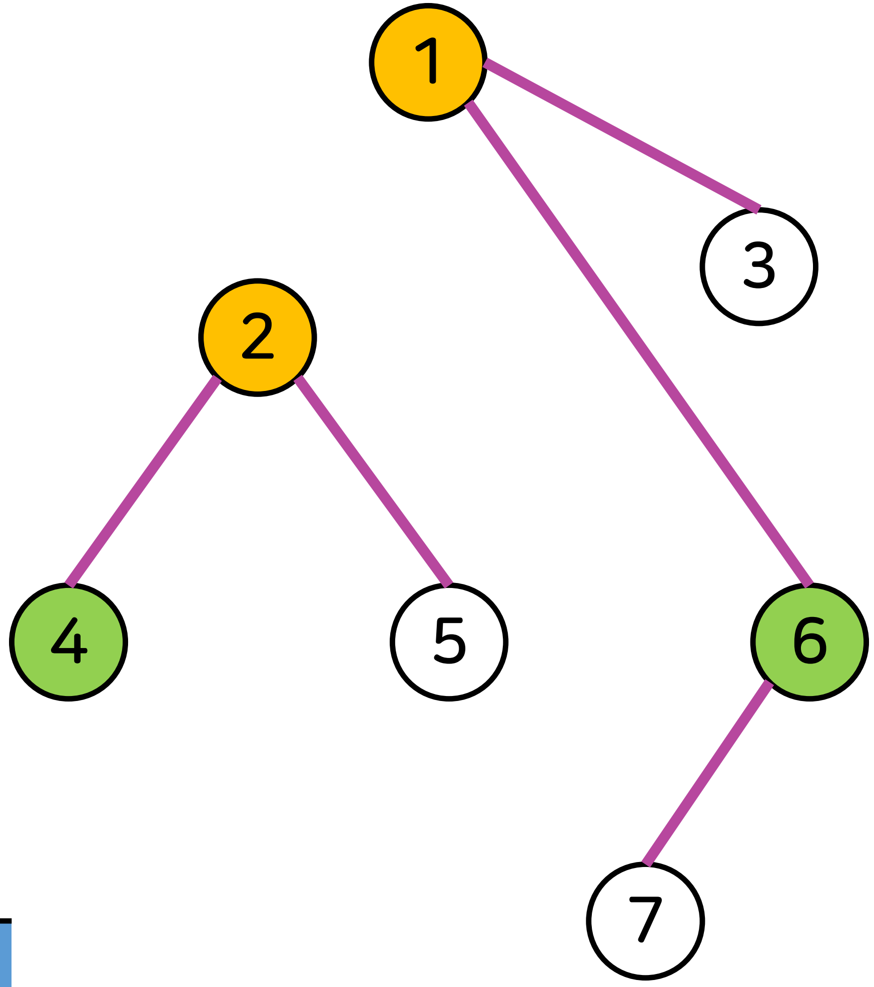


어떻게 동작할까? - UNION

노드 a	노드 b
6	4

집합 a	집합 b
1	2

노드	1	2	3	4	5	6	7
부모	1	2	1	2	2	1	6

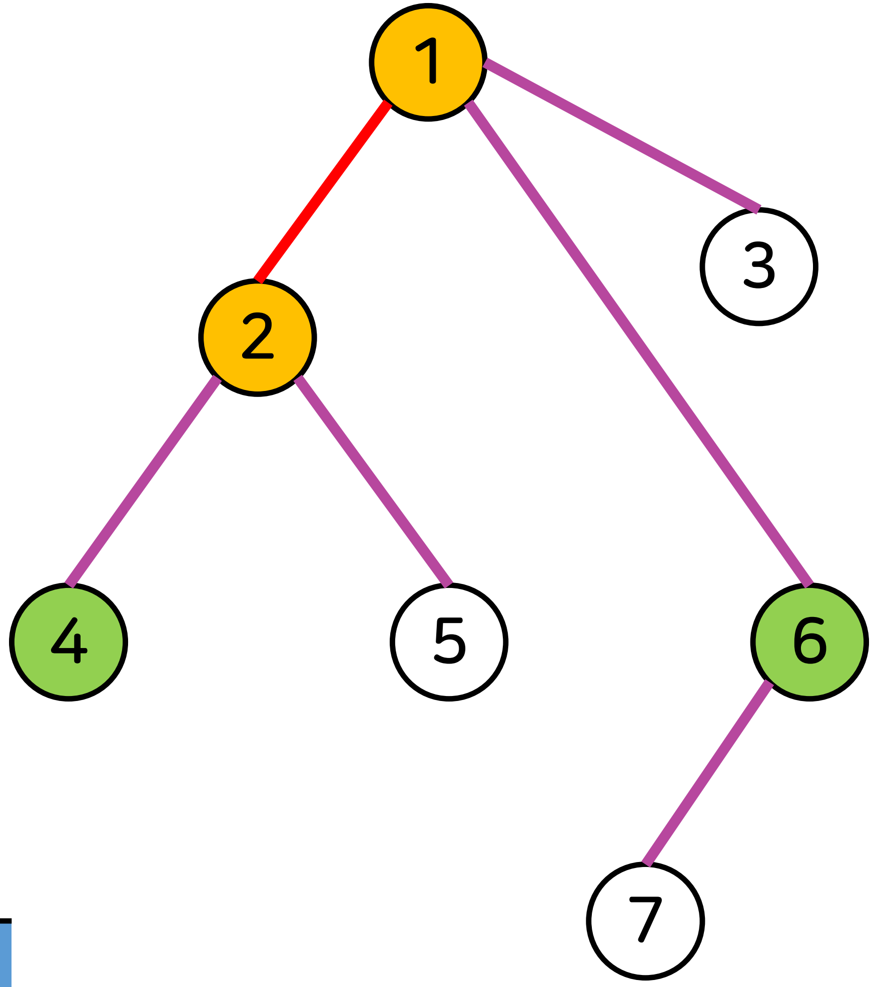


어떻게 동작할까? - UNION

노드 a	노드 b
6	4

집합 a	집합 b
1	2 -> 1

노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	1	6



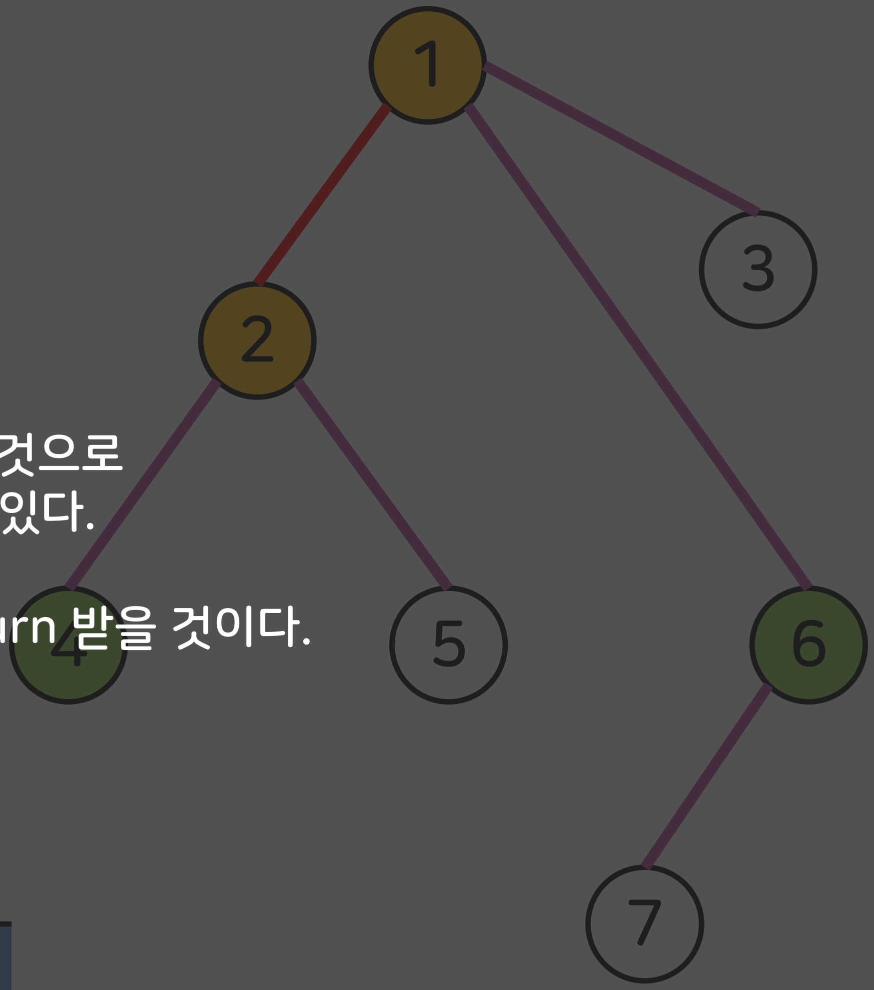
어떻게 동작할까? - UNION

노드 a	노드 b
6	4

집합 a	집합 b
1	2 -> 1

노드 2의 root를 노드 1로 설정하는 것으로
두 집합이 하나로 이어졌다고 볼 수 있다.

이후 FIND(4), FIND(5)를 수행할 시, 1을 return 받을 것이다.



노드	1	2	3	4	5	6	7
부모	1	1	1	2	2	1	6

연습문제들

- BOJ 1717 집합의 표현
- BOJ 1976 여행 가자
- BOJ 18116 로봇 조립
- BOJ 14595 동방 프로젝트 (Large)

SPOILER ALERT!

이 다음 슬라이드에는 문제 풀이가 적혀 있습니다.
본인 힘으로 문제를 풀어보고 싶은 사람들은
빠르게 스킵 or 문제를 다 풀어보고 넘어가시길 바랍니다.

BOJ 1717 집합의 표현

- 전형적인 Disjoint Set 문제이다.

- 입력 형식에 따라 아래와 같은 코드를 짜면 된다.
  - 0 a b
    a와 b를 UNION한다.

  - 1 a b
    a와 b가 속한 집합을 FIND를 통해 찾고, 그 둘이 같은지 비교한다.

BOJ 1976 여행 가자

- 역시나 전형적인 Disjoint Set 문제이다.
- 우선 주어진 인접 행렬을 가지고 연결된 도시를 UNION한다.
- 그리고 여행 계획에 포함된 도시들이 모두 같은 집합에 속하는지 FIND를 통해 판단한다.

  모두 같은 집합에 속하면 가능한 여행 계획, 하나라도 다른 집합에 속하면 불가능한 여행 계획이다.

BOJ 18116 로봇 조립

- 약간의 응용이 필요한 Disjoint Set 문제이다.
  기존의 Disjoint Set 문제가 "내가 속한 집합"을 찾는 문제였다면,
  이번에는 "내가 속한 집합의 크기"를 찾는 문제이다.

- 10^6개의 부품을 일일이 다 찾아보는 건 시간초과가 날 것이다.
  그렇다면 우리는 더 빠르게 집합의 크기를 구할 수 있어야 한다.

- 생각해보면 단순하다. UNION으로 집합 a, b를 합쳤다면,
  그 결과가 되는 집합의 크기는 (집합 a의 크기) + (집합 b의 크기)가 될 것이다.
  처음에 모든 집합의 크기는 1로 초기화하고, UNION 과정에서 집합 두개의 크기를 더해 주면 된다.

  자세한 코드는 다음 슬라이드를 참고하자.

BOJ 18116 로봇 조립

```
int parent[SIZE], cnt[SIZE]; //parent[i] = i, cnt[i] = 1로 모두 초기화 한다.

void UNION(int node_a, int node_b) {
	int root_a = FIND(node_a), root_b = FIND(node_b);
	//두 집합이 서로소인 경우에만,
	if (root_a != root_b) {
		//두 집합을 합치고, 크기 계산
		parent[root_b] = root_a;
		cnt[root_a] += cnt[root_b];
	}
}
```

BOJ 14595 동방 프로젝트 (Large)

- 조금은 특이한 문제이다.
  한번 방을 합칠 때마다 단순히 (y-x)회 UNION을 돌리면 당연히 시간 초과를 받는다.

- 그럼 어떻게 해야 할까?
  우리는 (또) UNION을 조금 바꿔서 짜야 한다.

- 우선 방들의 집합의 대표 원소는 무조건 가장 큰 번호가 맡도록 한다.
  그렇게 되면 FIND(a) >= a 가 보장된다.

- A부터 b까지의 방을 합친다는 것은, FIND(a)부터 FIND(b)까지의 방이 모두 대표 원소를 FIND(b)로 가지게 된다는 의미이다.

- 또한, 여기서 모든 방의 대표 원소를 바꿀 필요 없이, 기존에 집합의 대표 원소였던 방만 root를 FIND(b)로 만들어주면 된다. 이렇게만 해도 반복문의 실행 횟수는 크게 줄어든다.

BOJ 14595 동방 프로젝트 (Large)

```
int parent[SIZE]; //parent[i] = i로 초기화한다.

void UNION(int node_a, int node_b) {
        int root_a = FIND(node_a), root_b = FIND(node_b);
        //시작 : node_a를 포함하는 집합의 대표 번호
        //조건 : 지금 방이 root_b가 아닐 때까지(합쳐야 할 범위를 다 합쳤다면 종료)
        //다음 : 바로 옆 방을 포함하는 집합의 대표 번호
        for (int i = root_a; i != root_b; i = FIND(i + 1)) {
                parent[root_b] = root_a;
                cnt[root_a] += cnt[root_b];
        }
}
```

챕터 2: MST

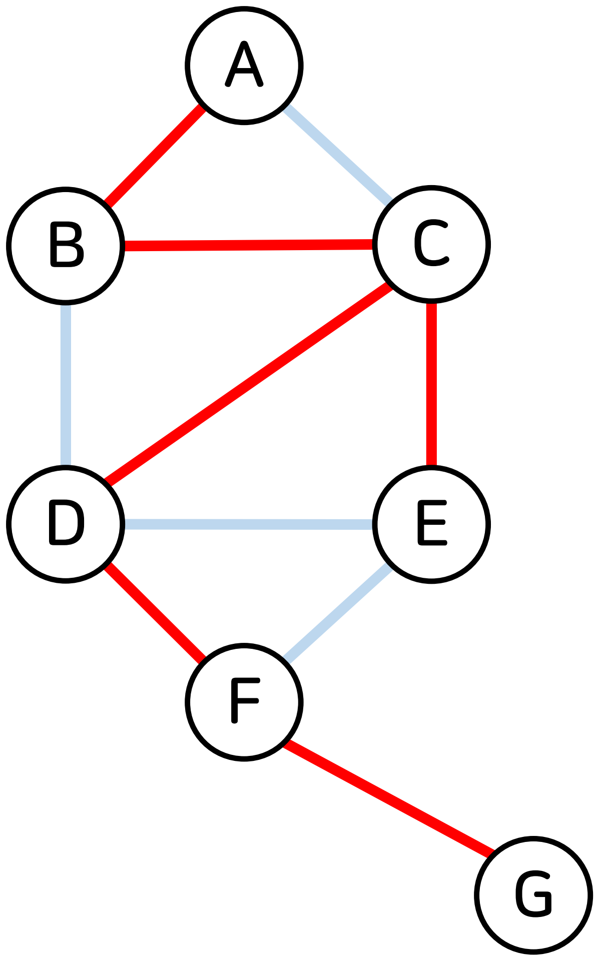
최소 스패닝 트리? Minimum Spanning Tree 알고리즘을 짜보자!

스패닝 트리?

- 그래프에 포함된 모든 정점을 포함하는 트리 구조

- 옆의 그래프에서 빨간 간선으로 이루어진 부분이 저 그래프의 스패닝 트리라고 할 수 있다.

- 당연히 스패닝 트리는 유일하지 않다.

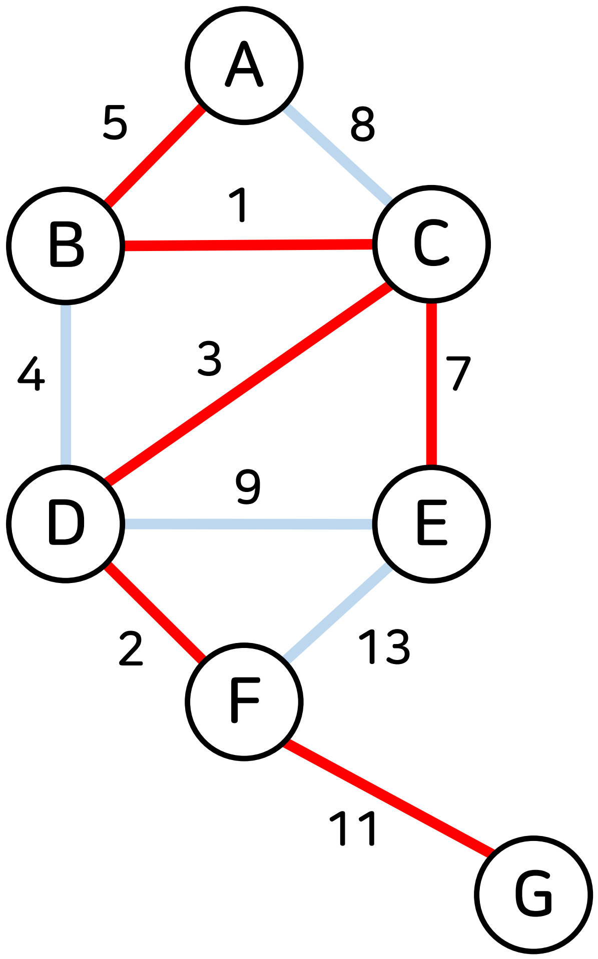


MST?

- 가중치 그래프에서,
  간선의 가중치 합이 가장 작은 스패닝 트리

- 옆의 그래프에서 빨간 간선으로 이루어진 부분이
  저 그래프의 MST라고 할 수 있다.

- 이 또한 유일하지는 않을 수 있다.



Kruskal 알고리즘

- 주어진 그래프에서 MST를 만드는 알고리즘

- 단계는 다음과 같다.
  - 그래프의 간선을 가중치 기준으로 오름차순 정렬한다.
  - 각 간선에 대하여 다음 작업을 수행한다.
    - 간선의 양 끝 정점이 속한 집합을 찾는다.
    - 다른 집합이라면 두 집합을 합치고,
해당 간선을 MST에 속한 간선으로 취급한다.
    - 같은 집합이라면 넘어간다.

Kruskal 알고리즘

- 주어진 그래프에서 MST를 만드는 알고리즘
- 단계는 다음과 같다.
  - 그래프의 간선을 가중치 기준으로 오름차순 정렬한다.
  - 각 간선에 대하여 다음 작업을 수행한다.
    - 간선의 양 끝 정점이 속한 집합을 찾는다.
    - 같은 집합이라면 두 집합을 합치고,
      해당 간선을 MST에 속한 간선으로 취급한다.
    - 다른 집합이라면 넘어간다.

음?
앞에서 비슷한 말을 본 것 같은데?

Kruskal 알고리즘

- 주어진 그래프에서 MST를 만드는 알고리즘

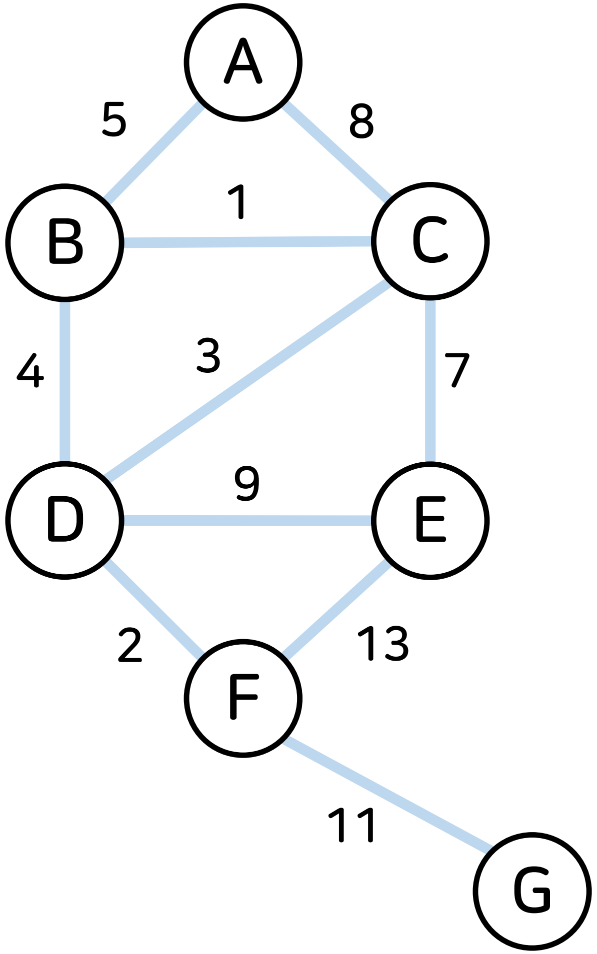
- 단계는 다음과 같다.
  - 그래프의 간선을 가중치 기준으로 오름차순 정렬한다.
  - 각 간선에 대하여 다음 작업을 수행한다.
    - 간선의 양 끝 정점이 속한 집합을 찾는다. -> Disjoint Set의 FIND
    - 같은 집합이라면 두 집합을 합치고, -> Disjoint Set의 UNION
      해당 간선을 MST에 속한 간선으로 취급한다.
    - 다른 집합이라면 넘어간다.

Kruskal 알고리즘도 Disjoint Set을 응용해서 구현한다는 것을 알 수 있다.

Kruskal 알고리즘

```
typedef pair<int, pair<int, int>> Edge;
priority_queue<Edge, vector<Edge>, greater<Edge>>edge; //min heap을 위해 이렇게 선언한다.
int ans;
int main() {
	//UNION과 FIND는 앞에서의 정의를 참고하고, 간선 입력 부분은 생략한다.
	while (!edge.empty()) {
		Edge e = edge.top(); edge.pop();
		if(FIND(e.second.first)!=FIND(e.second.second)) {
			UNION(e.second.first, e.second.second);
			ans += e.first;
		}
	}
}
```

어떻게 동작할까?

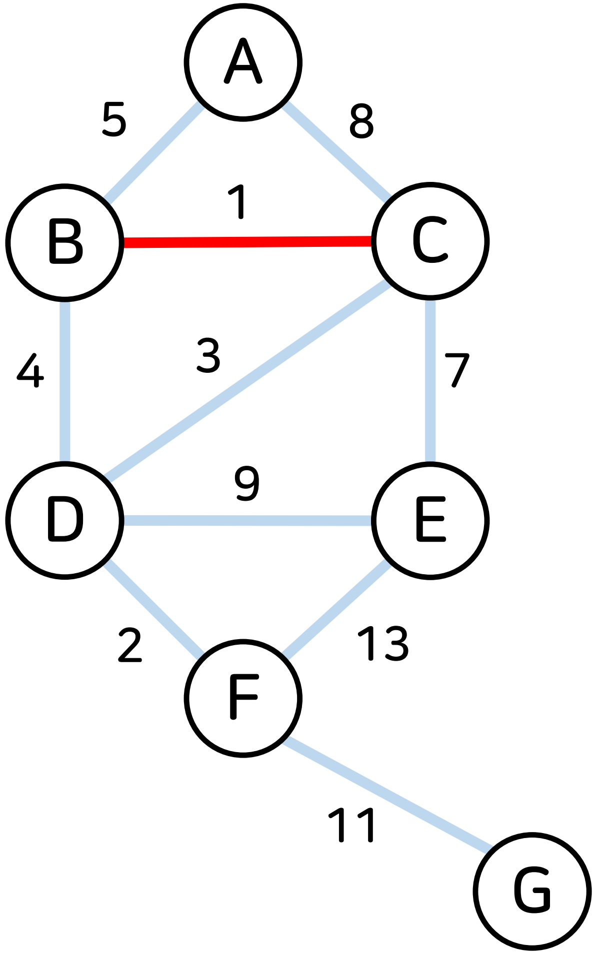


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

가중치 합
0

어떻게 동작할까?

B, C는 현재 다른 집합이므로
이 간선을 MST에 포함한다.

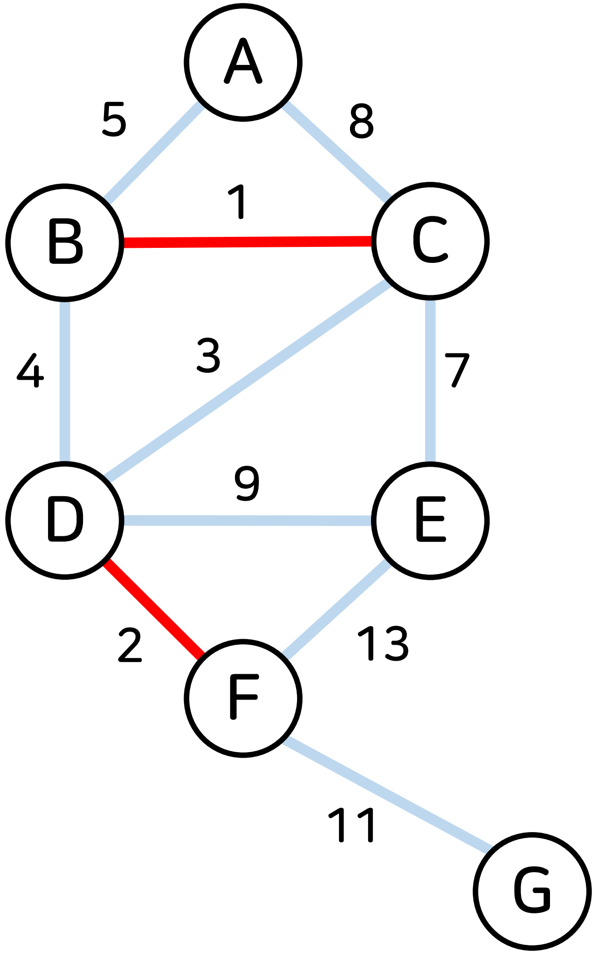


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

가중치 합
0 + 1

어떻게 동작할까?

D, F는 현재 다른 집합이므로
이 간선을 MST에 포함한다.



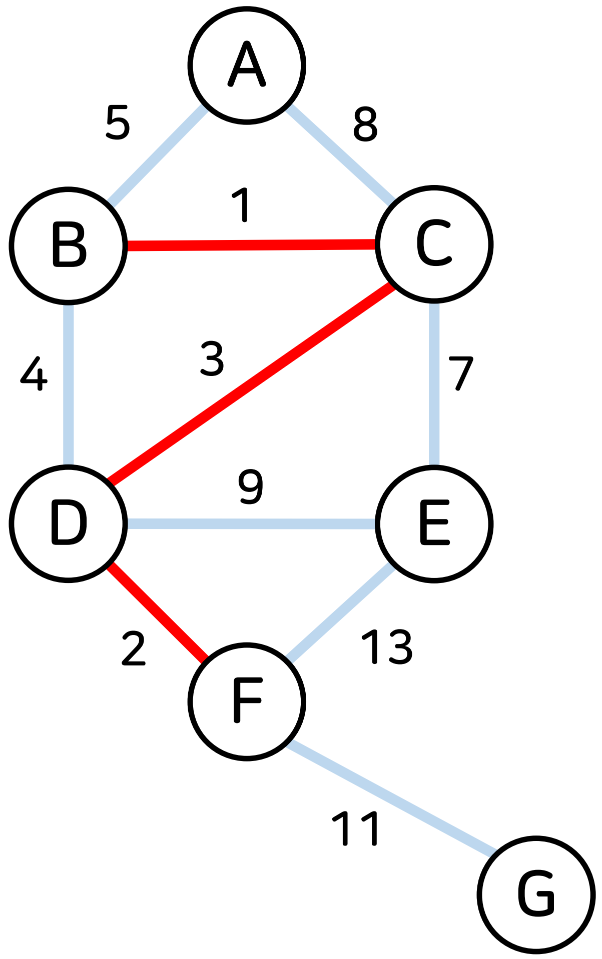
연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

가중치 합
1 + 2

어떻게 동작할까?

C, D는 현재 다른 집합이므로
이 간선을 MST에 포함한다.

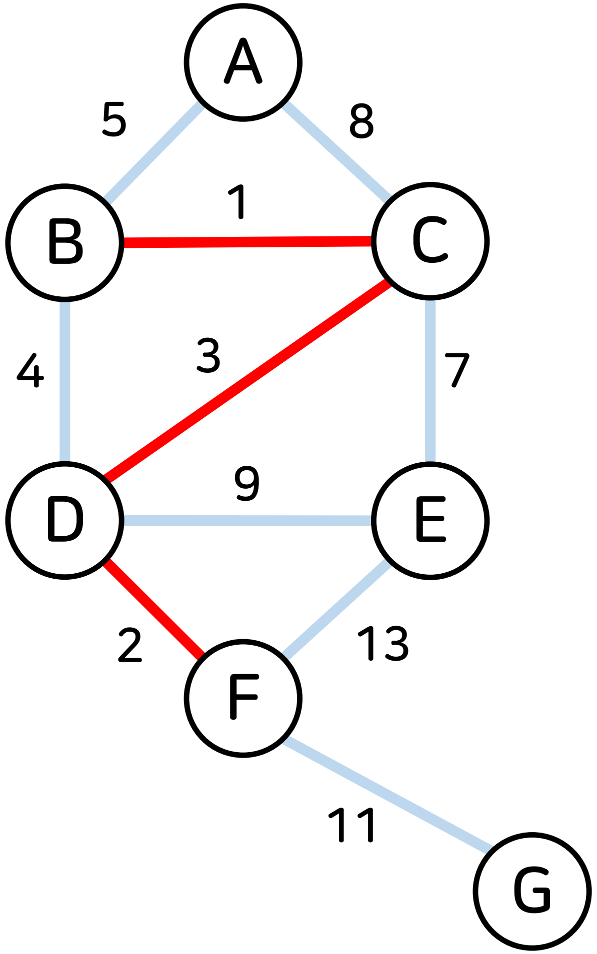
가중치 합
3 + 3



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?

B, D는 현재 같은 집합이므로
이 간선은 포함하지 않는다.



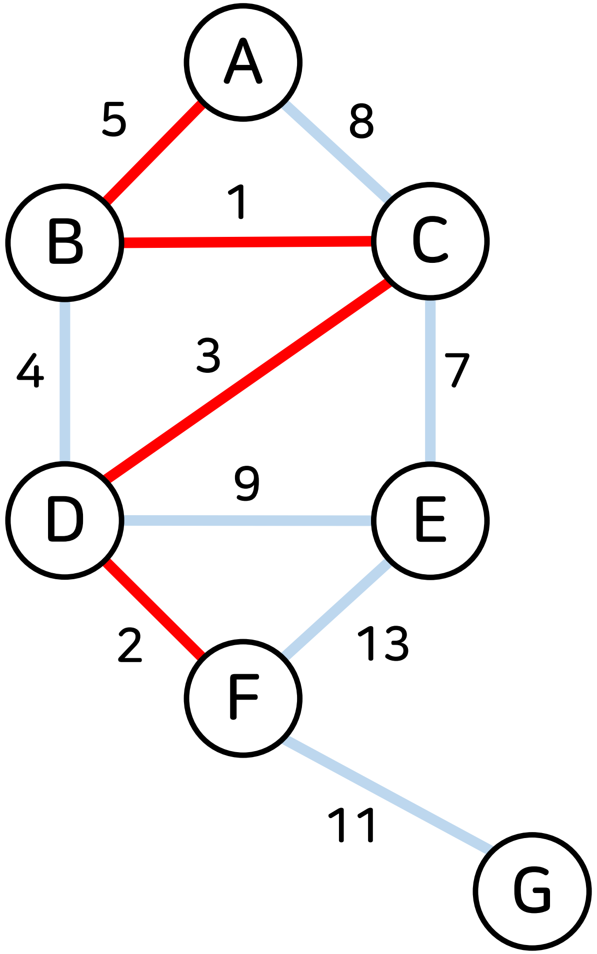
연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

가중치 합
6

어떻게 동작할까?

A, B는 현재 다른 집합이므로
이 간선을 MST에 포함한다.

가중치 합
6 + 5

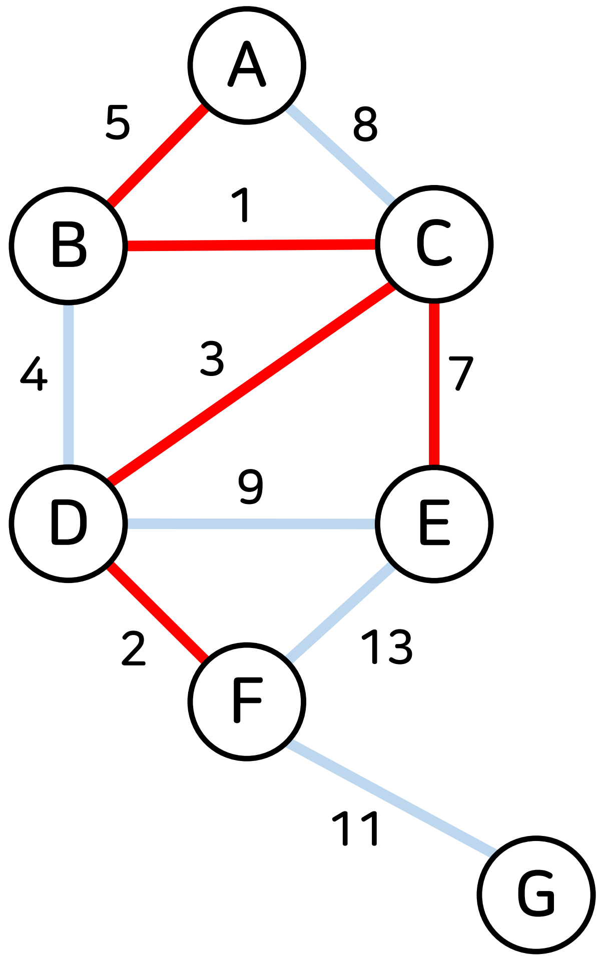


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

C, E는 현재 다른 집합이므로
이 간선을 MST에 포함한다.

가중치 합
11 + 7

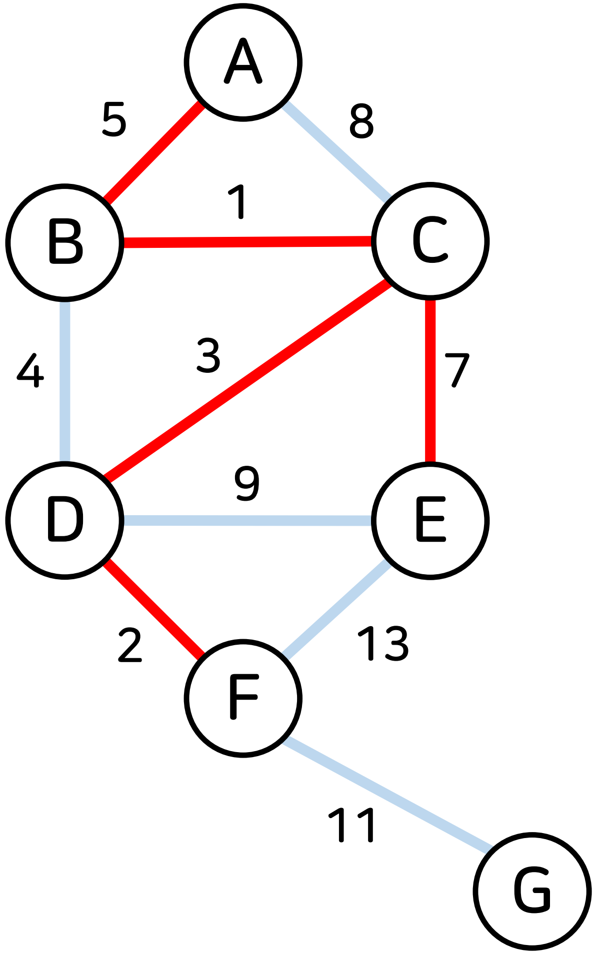


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

A, C는 현재 같은 집합이므로
이 간선은 포함하지 않는다.

가중치 합
18

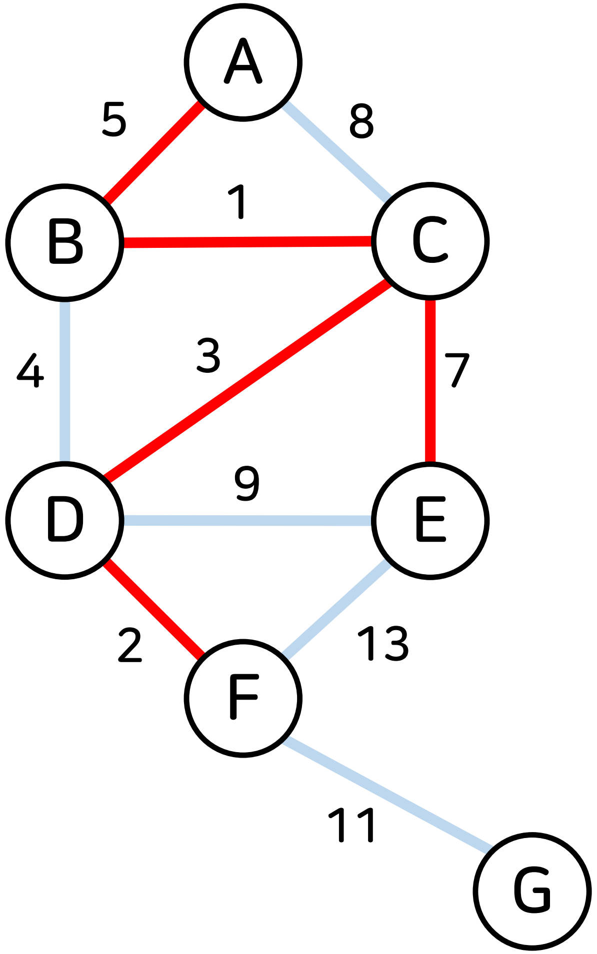


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

C, E는 현재 같은 집합이므로
이 간선은 포함하지 않는다.

가중치 합
18

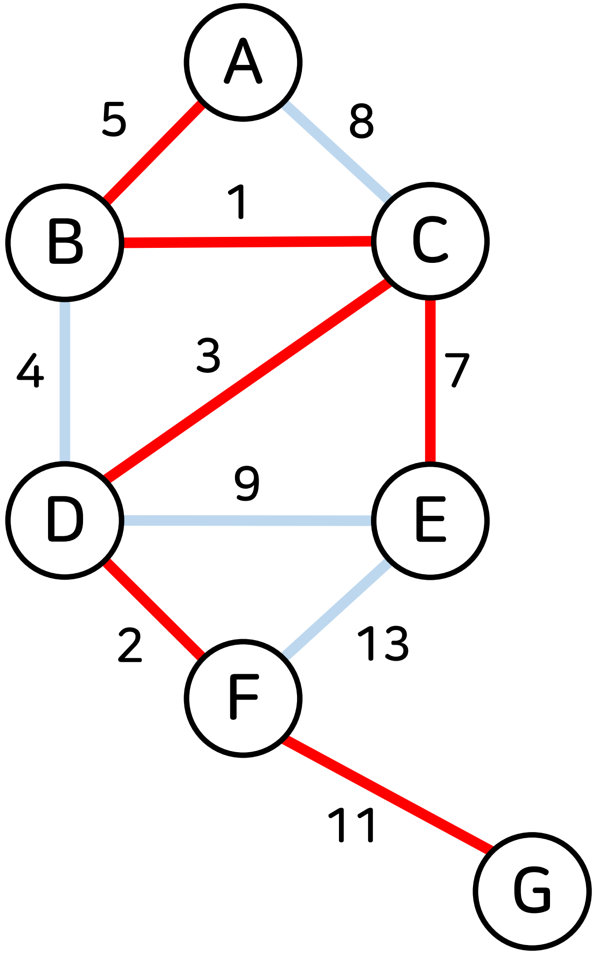


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

F, G는 현재 다른 집합이므로
이 간선을 MST에 포함한다.

가중치 합
18 + 11

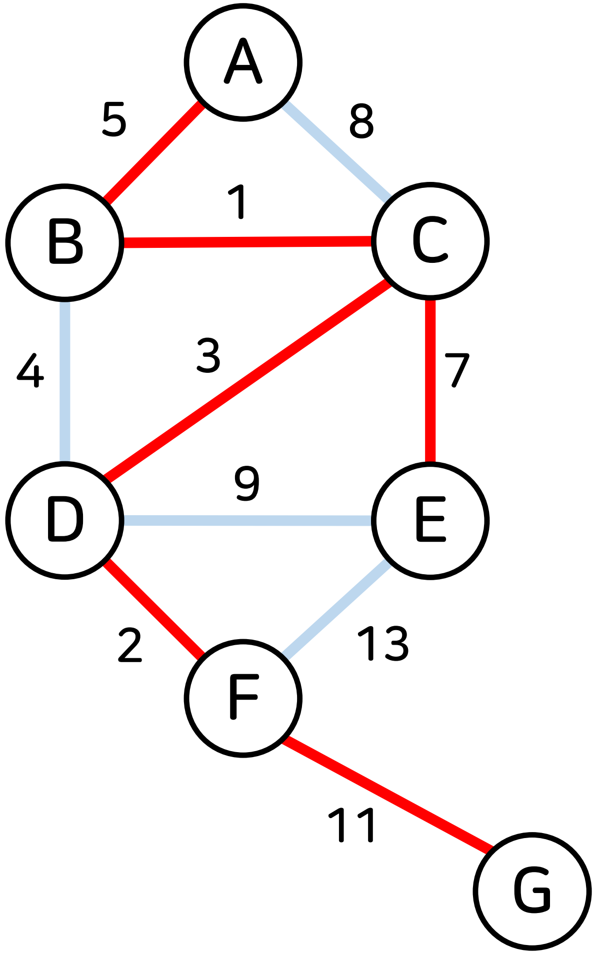


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

E, F는 현재 같은 집합이므로
이 간선은 포함하지 않는다.

가중치 합
29

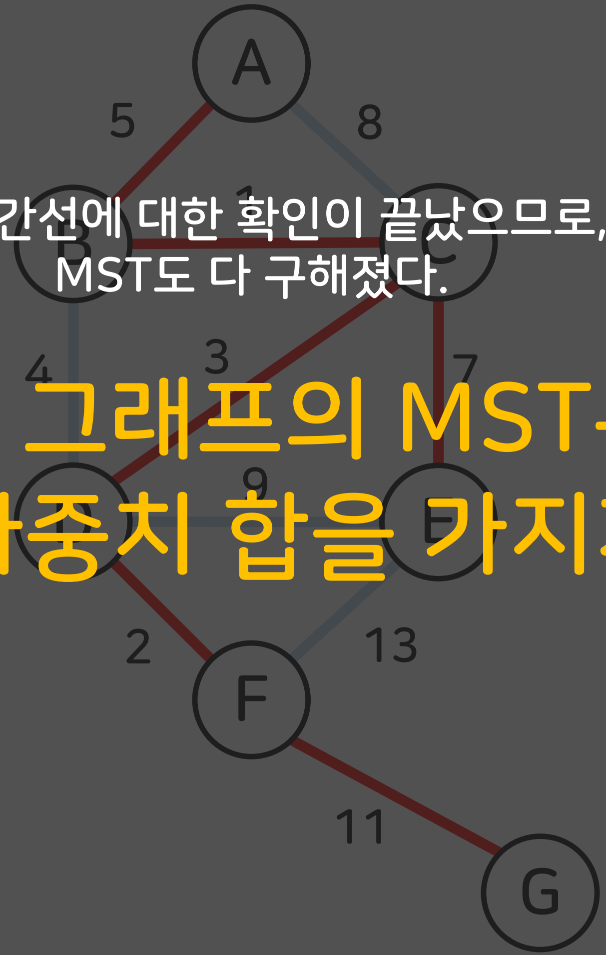


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

어떻게 동작할까?

E, F는 현재 같은 집합이므로
이 간선은 포함하지 않는다.

가중치 합
29



모든 간선에 대한 확인이 끝났으므로,
MST도 다 구해졌다.

현재 그래프의 MST는
총 29의 가중치 합을 가지게 된다.

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.
  - 임의의 정점 하나를 정하고, 그 정점을 MST에 포함한다.
  - 현재 MST에 포함되지 않고,
MST의 정점에 연결되어 있는 간선에 대해서 다음 과정을 수행한다.
    - 가장 가중치가 작은 간선 하나를 고르고,
그 간선이 MST와 MST 밖에 있는 정점을 연결한다면, 그 간선을 MST에 포함한다.

Prim 알고리즘

- 주어진 그래프에서 MST를 만드는 알고리즘

- 단계는 다음과 같다.
  - 임의의 정점 하나를 정하고, 그 정점을 MST에 포함한다.
  - 현재 MST에 포함되지 않고,
    MST의 정점이 연결되어 있는 간선에 대해서 다음 과정을 수행한다.
    - 가장 가중치가 작은 간선 하나를 보고,
      그 간선이 MST와 MST 밖에 있는 정점을 연결한다면, 그 간선을 MST에 포함한다.

설명이 조금 복잡하다...

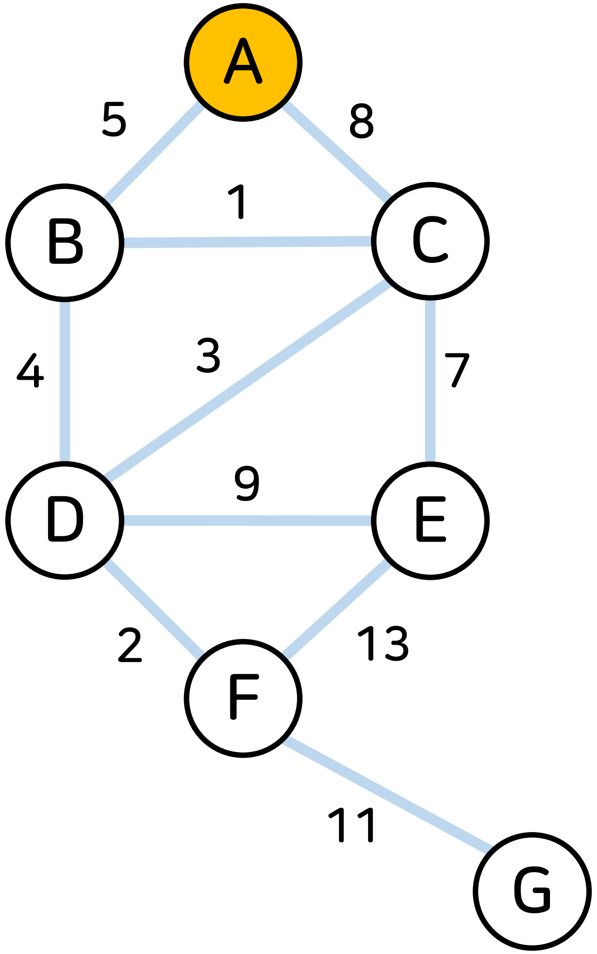
코드와 예시를 보면서 이해해 보자

Prim 알고리즘

```
priority_queue<Edge, vector<Edge>, greater<Edge>>edge; //MST에 연결된 간선의 집합, 가중치 순으로 정렬된다.
Vector<Edge>adj[SIZE]; //구현의 편의를 위해 인접 리스트에도 Edge 형식을 저장한다.
bool visit[SIZE]; //트리에 포함되었는지를 저장하는 배열이다.
int ans, start = 1; //시작 정점은 우선 1로 한다. 다른 번호로 해도 무방하다.
int main() {
        edge.push({ 0, {start, start} }); //시작 정점을 MST에 포함한다.
        while (!edge.empty()) {
                Edge now = edge.top(); //현재 집합 내에서 가장 가중치가 작은 간선
                edge.pop();
                if (visit[now.second.second]) continue; //이미 정점이 모두 MST에 포함되어 있다면 패스한다.
                visit[now.second.second] = true; //정점이 MST에 포함되었다고 표시한다.
                ans += now.first;
                //새로이 MST와 연결되는 간선들 모두에 대해서,
                for (Edge e : adj[now.second.second]) {
                        if (!visit[e.second.second]) edge.push(e); //아직 정점이 MST에 포함되지 않으면 집합에 push
                }
        }
}
```

어떻게 동작할까?

시작점은 A로 한다.

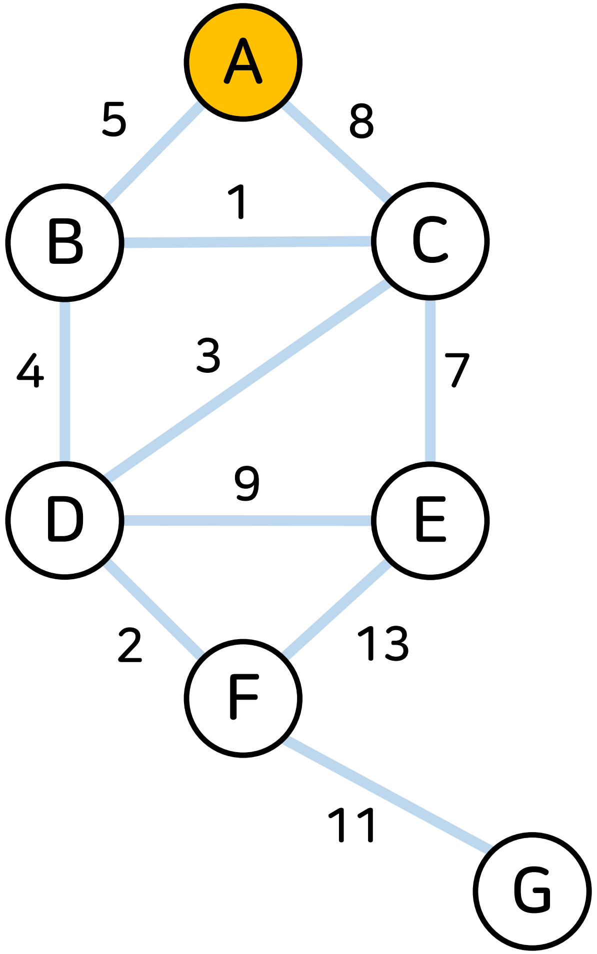


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

가중치 합
0

어떻게 동작할까?

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



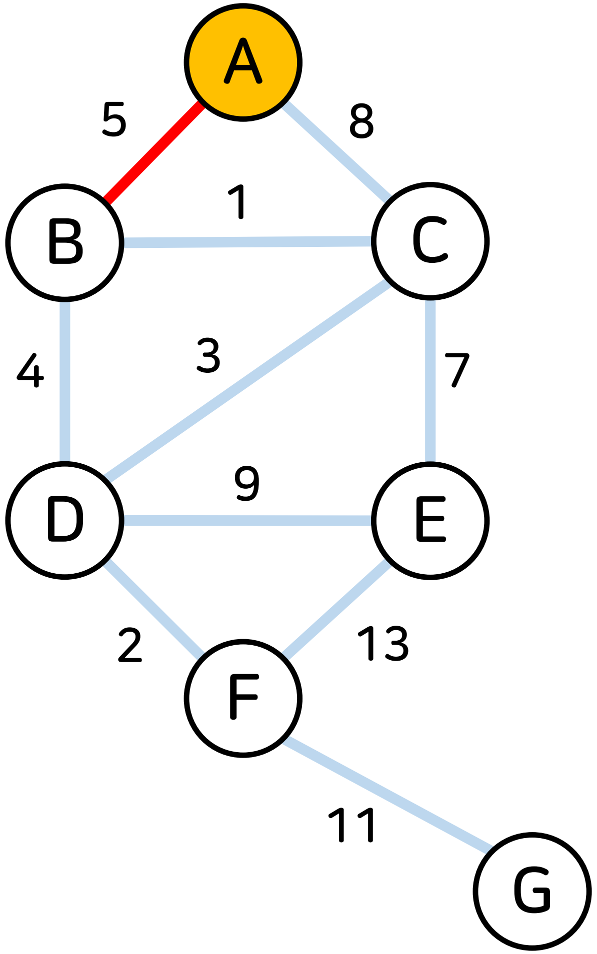
연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

가중치 합
0

어떻게 동작할까?

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

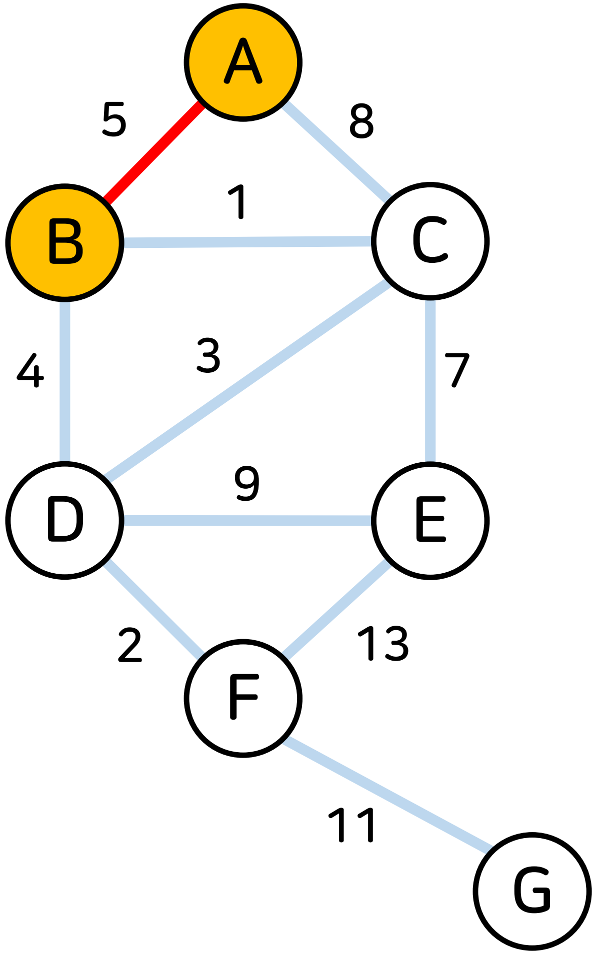
가중치 합
0 + 5



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.



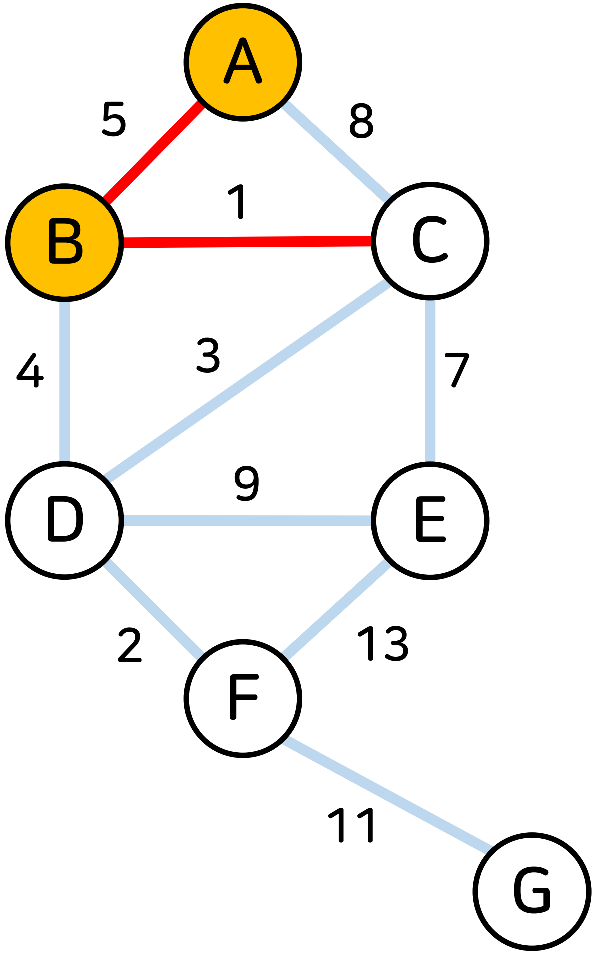
연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

가중치 합
5

어떻게 동작할까?

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

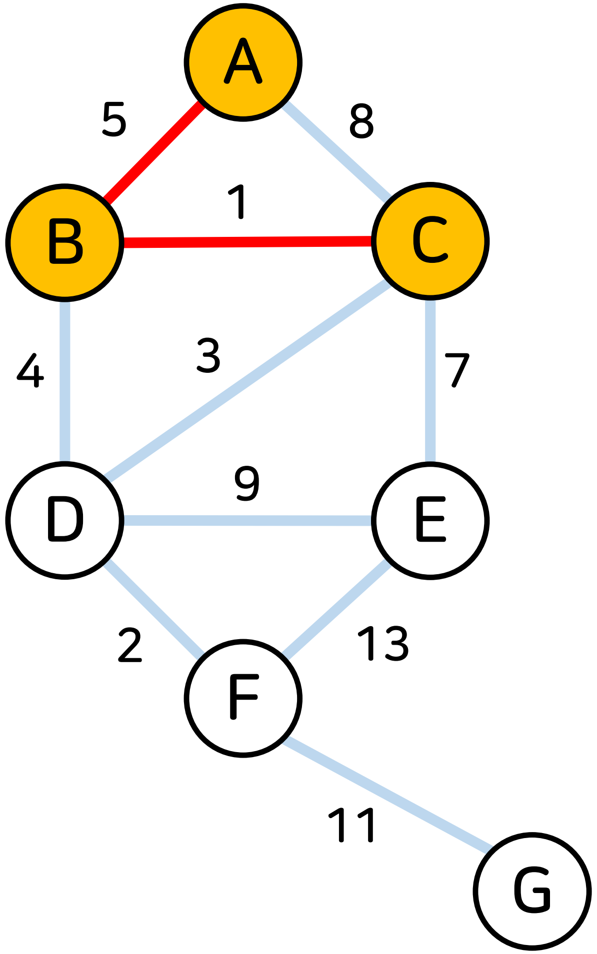
가중치 합
5 + 1



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.

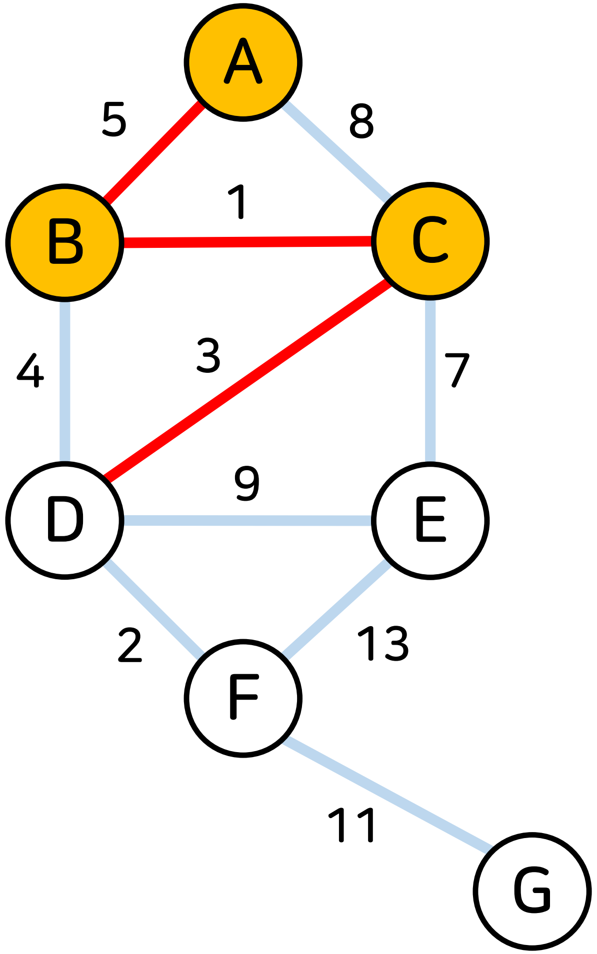


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

가중치 합
6

어떻게 동작할까?

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

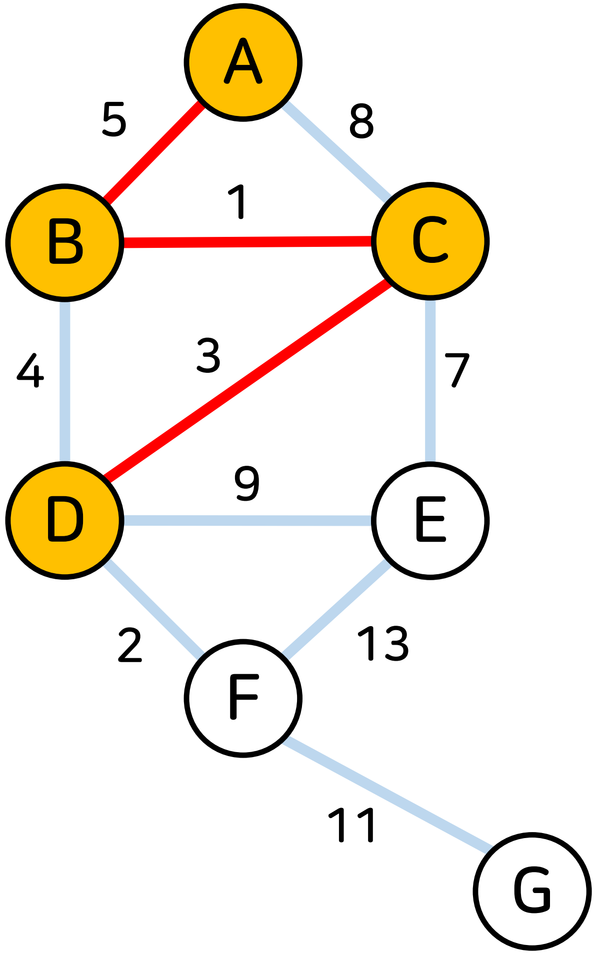


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

가중치 합
6 + 3

어떻게 동작할까?

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

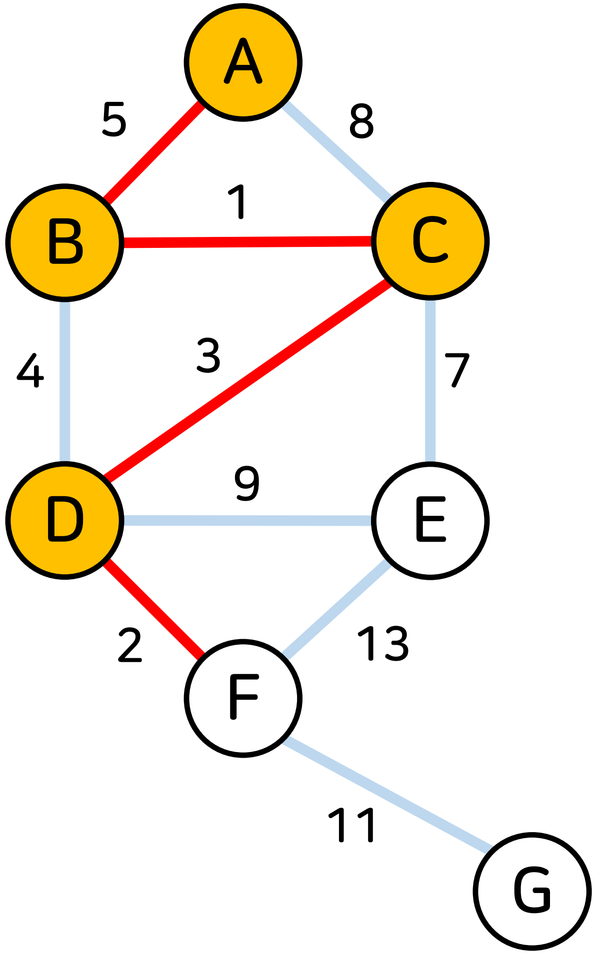


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

가중치 합
9

어떻게 동작할까?

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

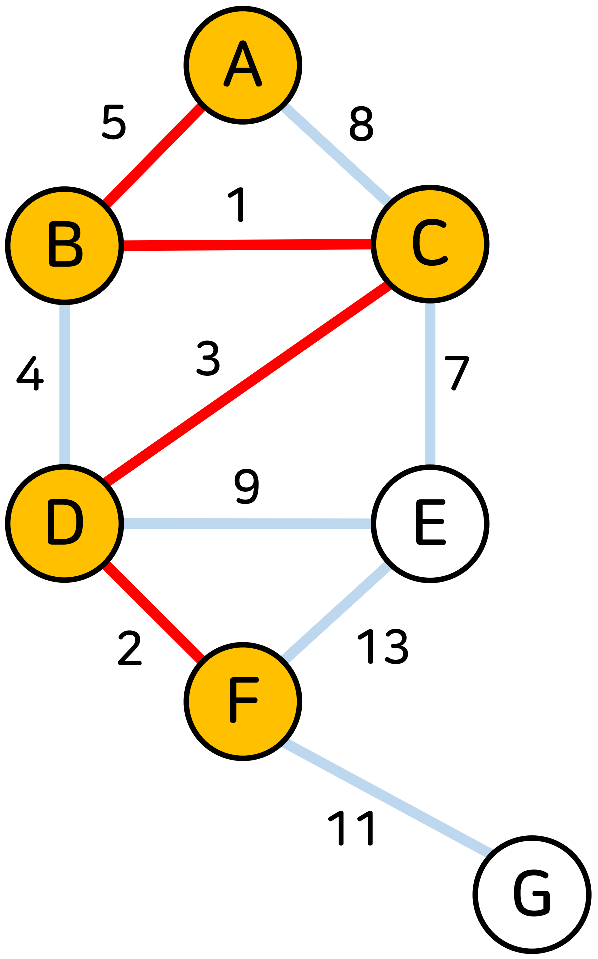


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

가중치 합
9 + 2

어떻게 동작할까?

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

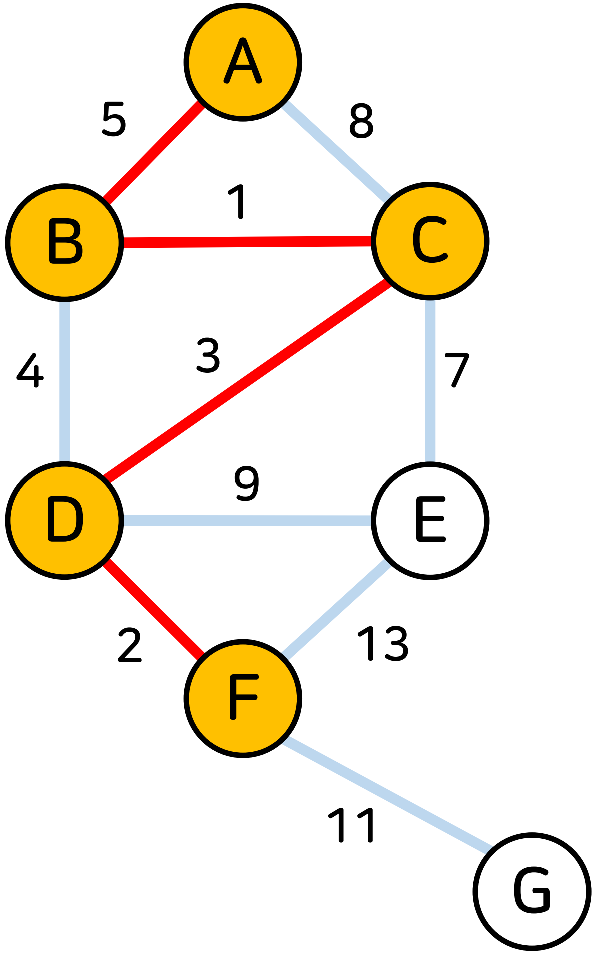


가중치 합
11

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.



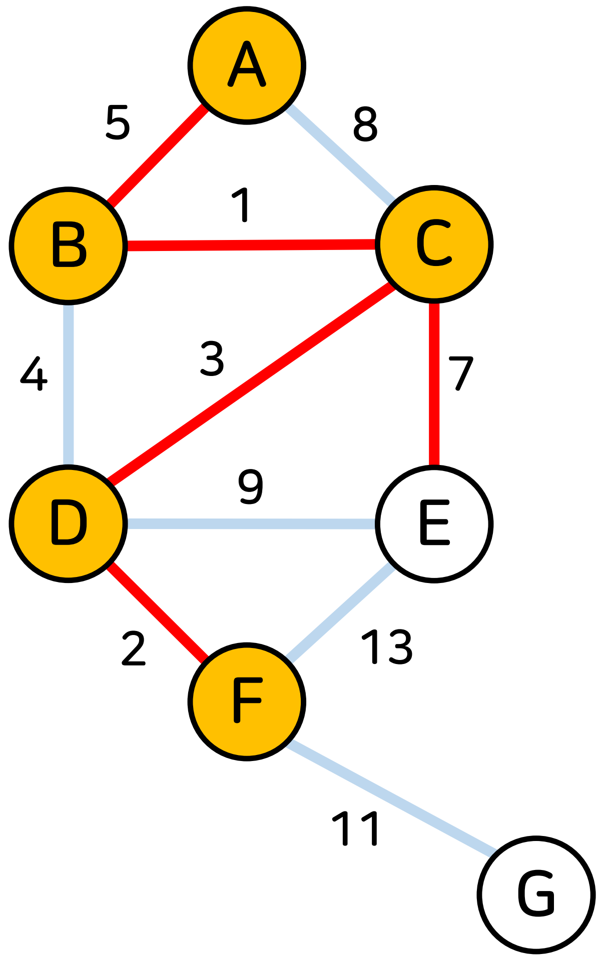
연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

가중치 합
11

어떻게 동작할까?

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

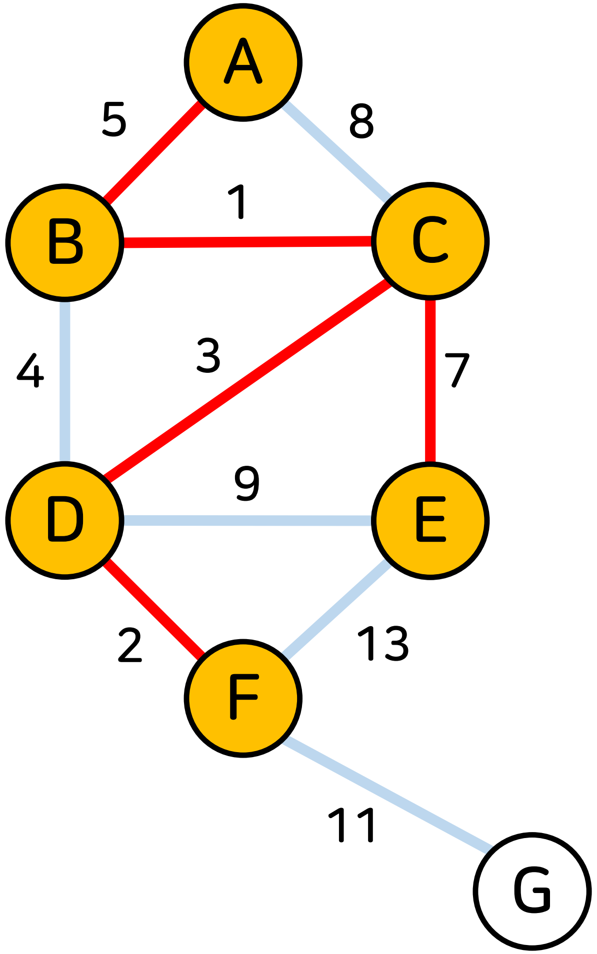
가중치 합
11 + 7



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.



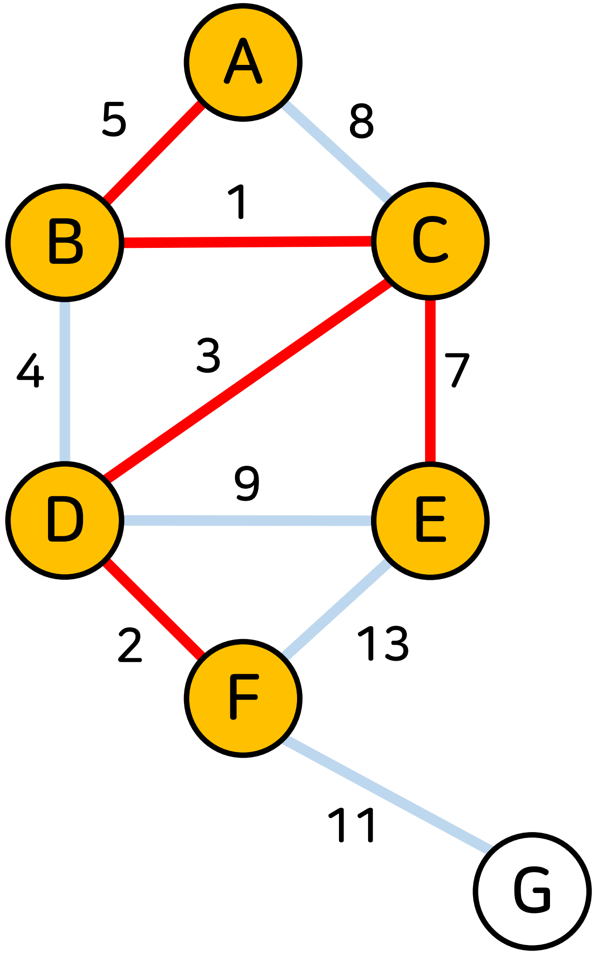
연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

가중치 합
18

어떻게 동작할까?

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

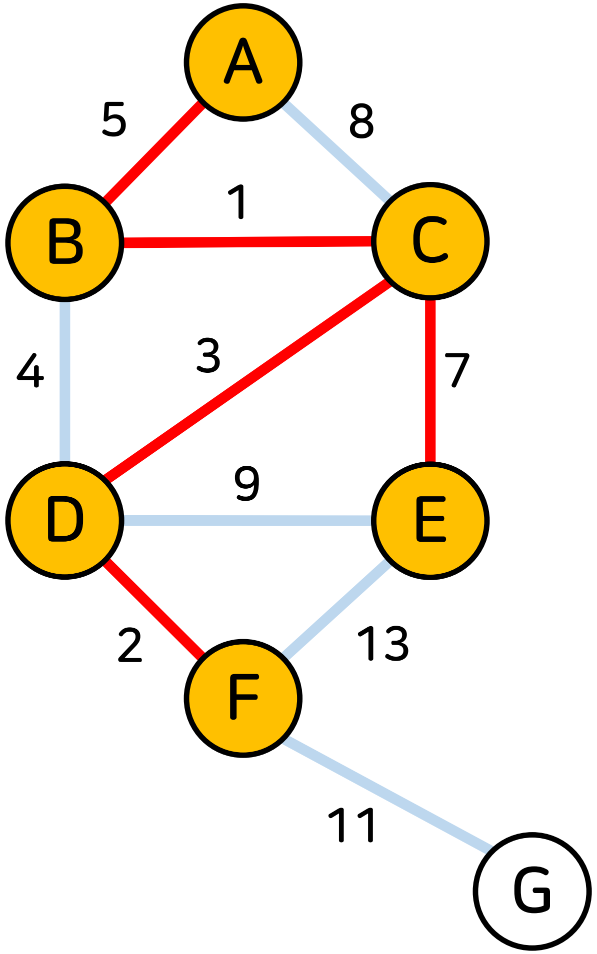
가중치 합
18



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.

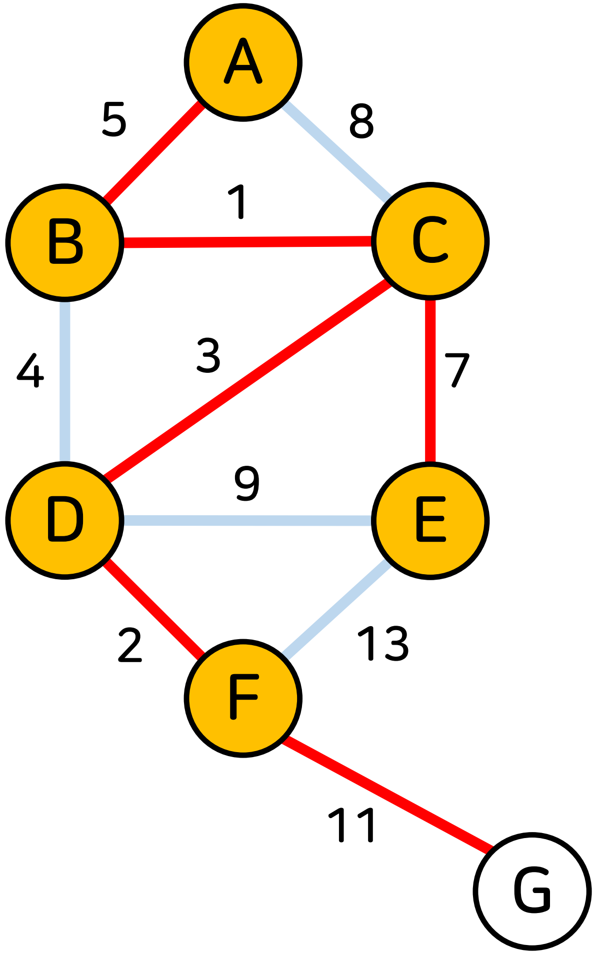


연결 정점	가중치
B, C	1
D, F	2
C, D	3
B, D	4
A, B	5
C, E	7
A, C	8
D, E	9
F, G	11
E, F	13

가중치 합
18

어떻게 동작할까?

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

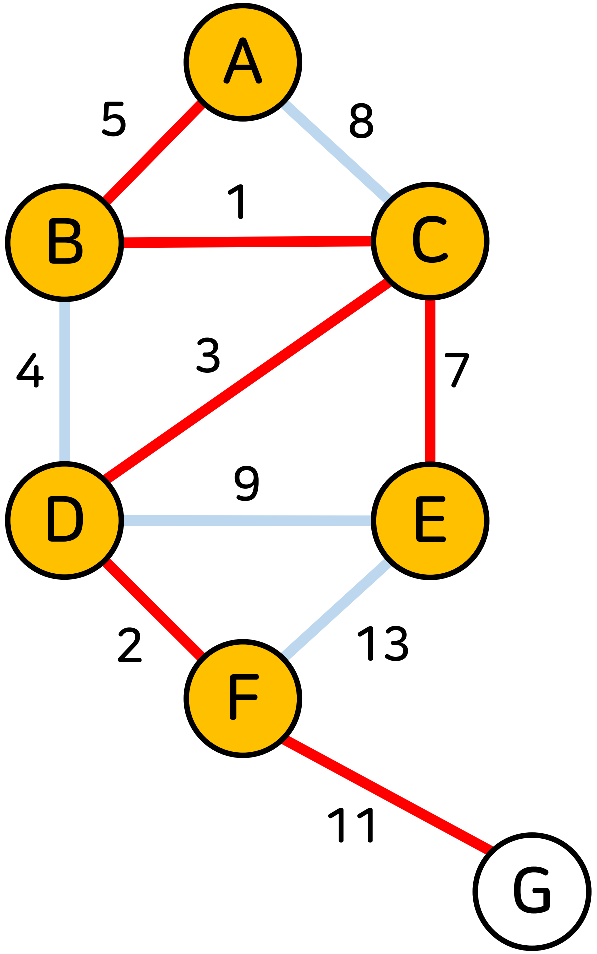


가중치 합
18 + 11

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.



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

가중치 합
29

어떻게 동작할까?

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

간선을 담아두는 집합이 비었으므로,
MST도 다 구해졌다.

현재 그래프의 MST는
총 29의 가중치 합을 가지게 된다.

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

- 분명 둘 다 MST를 구하는 알고리즘이다.
  그렇지만 뭔가 논리라던지, 실제로 전개되는 단계가 뭔가 다르다.

- 그럼 시간복잡도의 차이가 있지 않을까?

Kruskal vs. Prim

- 분명 둘 다 MST를 구하는 알고리즘이다.
  그렇지만 뭔가 논리라던지, 실제로 전개되는 단계가 뭔가 다르다.

결론부터 말하자면,

- 그럼 시간복잡도의 차이가 있지 않을까?

차이가 없다.

Kruskal vs. Prim

- 우선 Kruskal의 시간복잡도를 구해보자.

- Kruskal은 우선 간선을 모두 정렬해야 한다.
  -> O(E logE) = O(E log(V^2)) = O(E logV)

- 모든 간선에 대해서 이제 Find와 Union을 수행한다.
  Path Compression 등으로 Find 연산을 최적화하면 사실상 O(logV) 가 된다.
  이를 E번 반복하므로
  -> E * O(logV) = O(E logV)

- 총 시간복잡도는 O(E logV)가 된다.

Kruskal vs. Prim

- 다음은 Prim의 시간복잡도를 구해보자.

- 우선 최대 E번 priority queue에 간선 정보를 pop하게 된다.
  -> E * O(logE) = E* O(log(V^2) = E* O(logV) = O(E logV)

- 인접 리스트를 통해서 간선 정보를 priority queue에 push하는 과정은 최대 E번 발생한다.
  -> E * O(logE) = E* O(log(V^2) = E* O(logV) = O(E logV)

- 총 시간복잡도는 O(E logV)가 된다.

Kruskal vs. Prim

- 다음은 Prim의 시간복잡도를 구해보자.
- 우선 최대 E번 priority queue에 간선 정보를 pop하게 된다.
  -> E * O(logE) = E* O(log(V^2)) = E* O(logV) = O(E logV)
- 인접 리스트를 통해서 간선 정보를 priority queue에 push하는 과정은 최대 E번 발생한다.
  -> E * O(logE) = E* O(log(V^2)) = E* O(logV) = O(E logV)
- 총 시간복잡도는 O(E logV)가 된다.

시간복잡도가 같다고 성능이 완전히 같다고 말하기는 어렵지만,
대략적으로는 비슷하다는 것을 알 수 있다.
어느 알고리즘을 선택할지는 취향 문제이다.

연습문제들

- BOJ 1197 최소 스패닝 트리
- BOJ 1647 도시 분할 계획
- BOJ 14621 나만 안되는 연애
- BOJ 1045 도로

SPOILER ALERT!

이 다음 슬라이드에는 문제 풀이가 적혀 있습니다.
본인 힘으로 문제를 풀어보고 싶은 사람들은
빠르게 스킵 or 문제를 다 풀어보고 넘어가시길 바랍니다.

BOJ 1197 최소 스패닝 트리

- 이름부터 전형적인 MST 문제이다.
- 입력 받은 간선 정보를 가중치 기준으로 정렬하고, Kruskal 혹은 Prim 알고리즘을 돌리면서 MST에 포함된 간선의 가중치를 별도의 변수에 계속 더해주면 된다.

BOJ 1647 도시 분할 계획

- 언뜻 보면 MST라 생각하기 어렵지만, 여기서 우리는 트리의 성질을 생각해야 한다.

  트리는 어느 간선이든 하나 지우면 두 개의 트리로 나누어진다.

- 우리는 두 개의 트리를 두 개의 마을로도 생각할 수 있다.

- MST를 구하면서,
  MST에 포함된 간선 중 가중치가 가장 큰 간선을 저장해 둔다.
  (후술할 때 M이라고 하자.)

- 구해진 MST의 가중치 합에서 M을 빼주면, MST는 두개로 나누어지고, 이것이 문제에서 요구하는 답이 된다.

BOJ 14621 나만 안되는 연애

- MST 알고리즘은 그대로지만, 간선을 걸러서 저장해야 하는 문제이다.
- 간선을 입력받으면서 간선이 연결하는 대학이 남초/여초가 맞는지 확인한다.
  맞다면 저장하고, 아니면 우리에겐 필요없는 간선이다.
- 이렇게 입력받는 간선 정보 중 필요한 정보만 저장했다면 다음은 Kruskal 알고리즘을 돌리기만 하면 된다.
- 알고리즘 연산 후, 모든 정점이 같은 집합에 있는지 꼭 확인하자.

BOJ 1045 도로

- MST 문제이지만 가중치가 없다.
  여기서 우리는 문제에 명시된 '우선순위'를 가지고 간선을 정렬한다.

- 또한 모든 도시가 연결되어야 하므로 일단은 MST를 구해야 한다.
  M이 N-1 이상이라는 것이 보장되므로 MST 알고리즘을 쓰지 못할 이유는 없다.

- 그러나 간선을 M개 포함해야 하므로,
  MST를 구하고 남은 간선 중 우선순위가 높은 간선 M-(N-1)개의 가중치를 더해주자.
  이는 MST를 구하면서 포함되지 못한 간선을 따로 저장해두면 쉽게 할 수 있다.

- 단, 그래프가 연결 그래프라는 언급은 없으므로, 구성한 그래프가 과연 연결 그래프인지는 체크해야 한다.
  즉, 모든 도시가 같은 집합에 속하는지 확인해야 한다.

수고했어요!

혹시 위의 문제가 너무 쉬웠나요? 그럼…

- BOJ 17490 일감호에 다리 놓기
- BOJ 13344 Chess Tournament
- BOJ 17132 두더지가 정보섬에 올라온 이유
- BOJ 6416 트리인가?