SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

삼성 청년 SW 아카데미

SW문제해결응용

<알림>

본 강의는 삼성 청년 SW아카데미의 컨텐츠로
보안서약서에 의거하여
강의 내용을 어떠한 사유로도 임의로 복사,
촬영, 녹음, 복제, 보관, 전송하거나
허가 받지 않은 저장매체를
이용한 보관, 제3자에게 누설, 공개,
또는 사용하는 등의 행위를 금합니다.

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

Day4. Union Find/MST

SAMSUNG SW ACADEMY FOR YOUTH

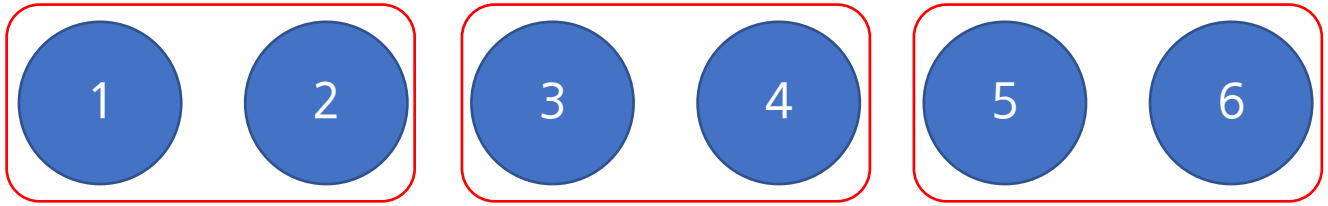
함께가요 미래로!
Enabling People

Union Find

Union Find : 합치고 찾는다.

- 서로소 집합을 관리하는 효율적인 자료구조 → **서로소 집합 자료구조**라고 한다.
- 서로소 집합 : 공통 원소가 없는 두 집합

- Union(1,2) : 1,2 를 합친다.
- Union(3,4) : 3,4 를 합친다.
- Union(5,6) : 5,6 을 합친다.
- 현재 그룹은 3개이다.

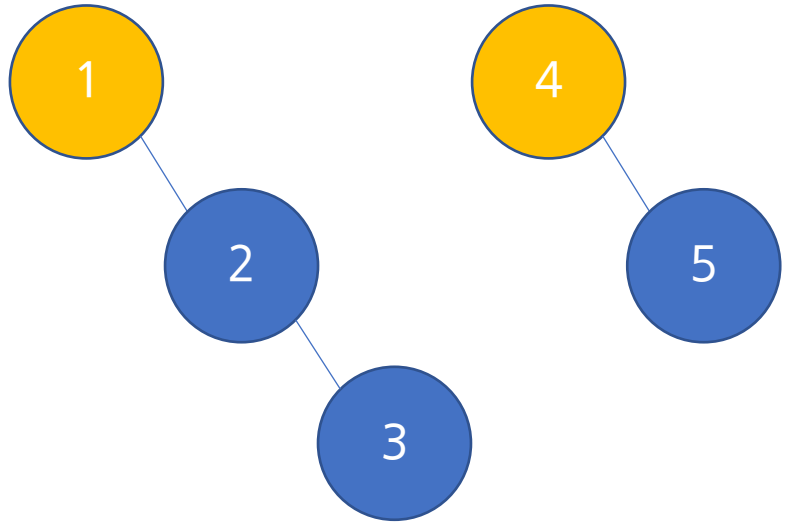


- Union Find 는 데이터 단위가 그룹이다.

Union Find 자세히 알아보기

Union() 은 그룹을 합친다.

- Union(1,2) : 2를 1의 자식 노드로 합친다.
- Union(2,3) : 3을 2의 자식 노드로 합친다.
- Union(4,5) : 5를 4의 자식 노드로 합친다.
- Union은 트리 구조로 이루어져 있다.



Find() 는 그룹을 찾는다.

- Find(3) : 3이 속한 그룹의 부모 노드를 반환한다. → 1
- Find(5) : 5가 속한 그룹의 부모 노드를 반환한다. → 4

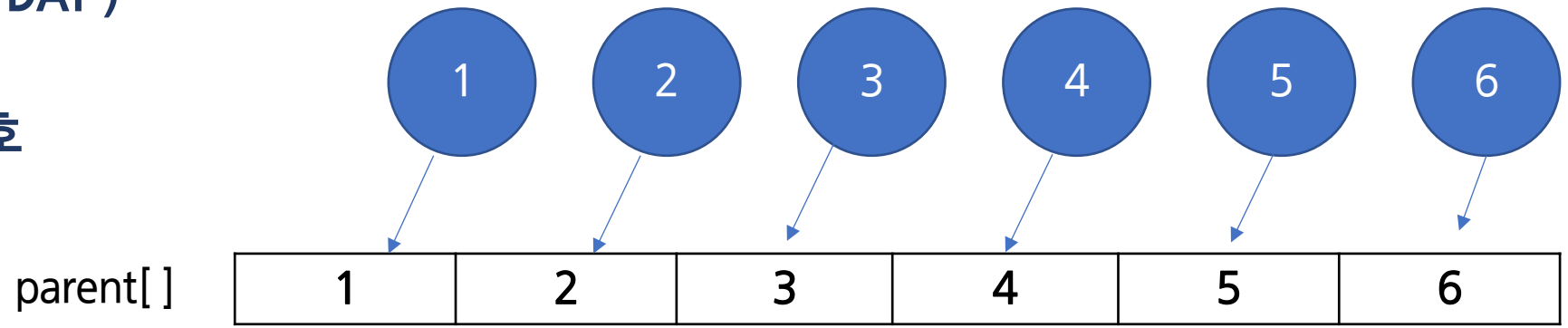
실생활이나 사회에서 그룹 단위로 관리되는 경우가 많아,

Union Find 관련 문제도 다양하게 출제된다.

- **최소 신장 트리 (MST)**
- **네트워크 연결성**
- **동적 연결성**
- **2-SAT**
- **등등**

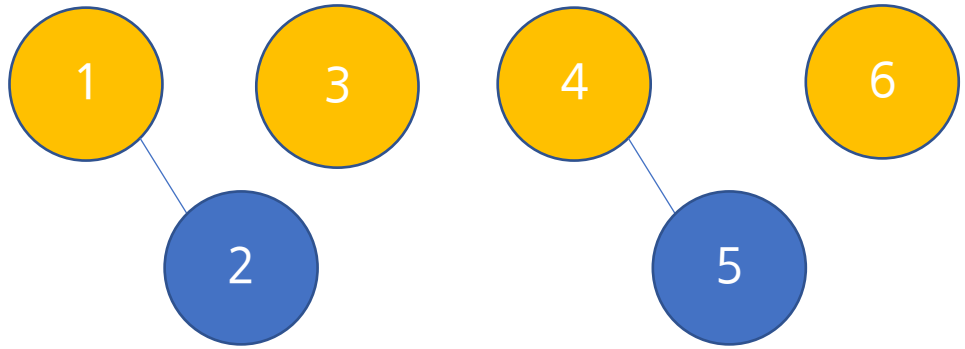
1. 초기 세팅 : 모두가 대표

- parent[ ] 배열 생성 (DAT)
- index : 노드 번호
- value : 대표 노드 번호



2. Union (그룹 합치기)

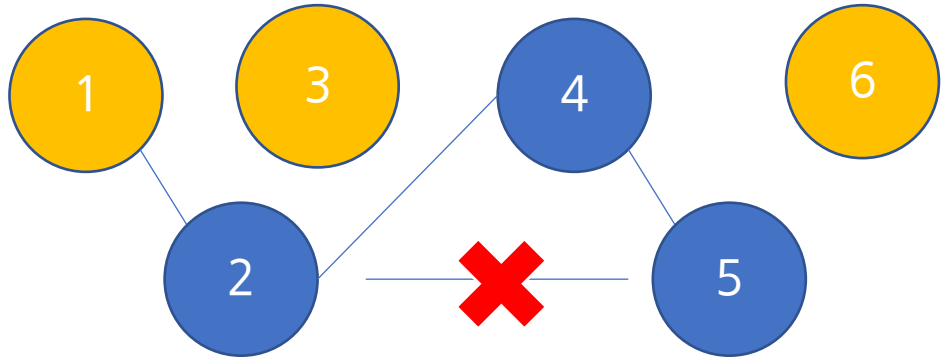
- **Union(1,2) : 2를 1의 자식 노드로 합친다.**
- **Union(4,5) : 5를 4의 자식 노드로 합친다.**
  - parent[] 의 값이 변경된다.



parent[ ]

1	1	3	4	4	6

- **Union(2,5) : 5를 2의 자식 노드로 합친다.**
  - 5의 대표가 2가 되는 것이 아니다.
- **Union(2,5) == Union(1,4) 가 되는 것**

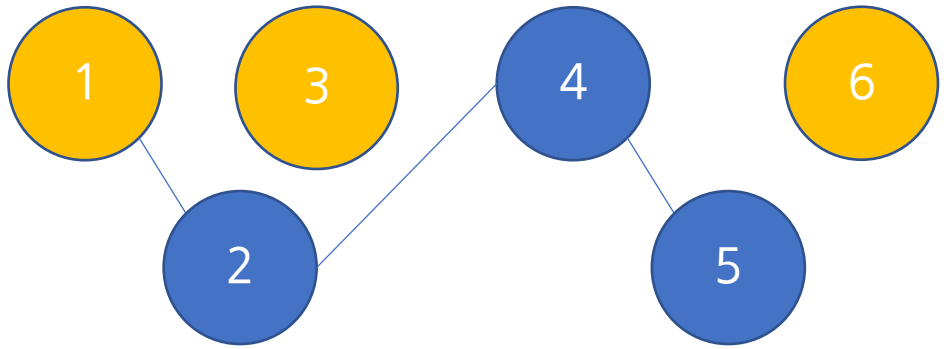


parent[ ]

1	1	3	1	4	6

3. Find(x) : x의 대표 노드 찾기

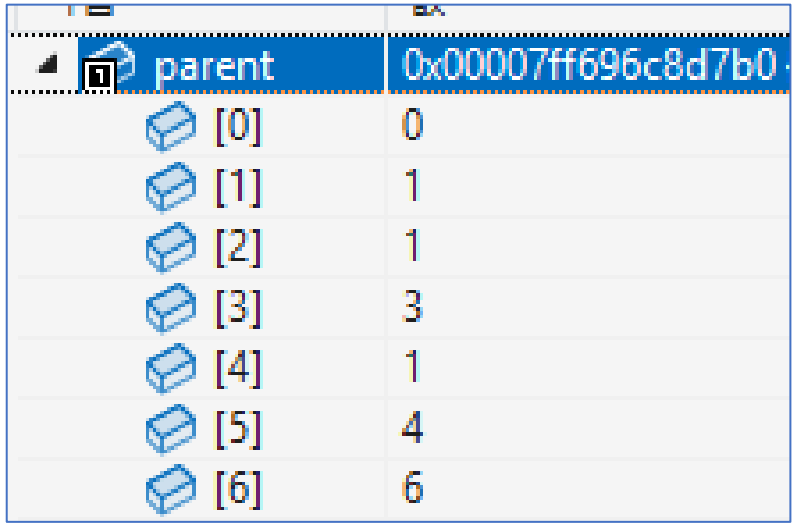
- **Find(1) : 1의 대표 노드 → 1**
- **Find(2) : 2의 대표 노드 → 1**
  - Find(parent[2])
- **Find(5) : 5의 대표 노드 → 4**
  4의 대표 노드 → 1
  - Find(parent[5]) → Find(parent[4]) → 1
  - 재귀적으로 동작한다.



parent[ ]

1	1	3	1	4	6

디버깅해서 parent[] 를 확인한다.



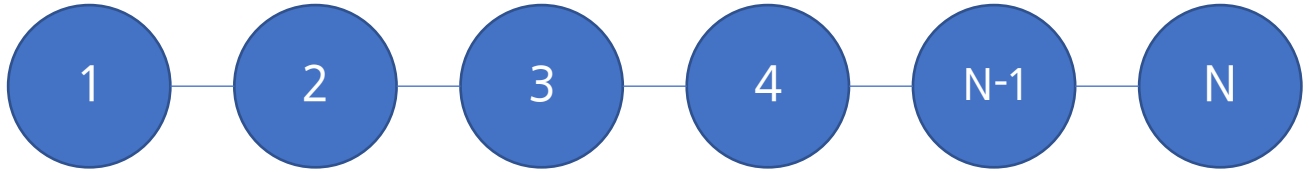

```
//Find 특정 그룹의 대표 찾기
int Find(int n) {
    //대표가 본인인 경우 return
    if (n == parent[n]) return n;
    //root 노드까지 찾아가기
    int root = Find( parent[n] );
    //대표 반환
    return root;
}
//Union 그룹을 합친다.
void Union(int A, int B) {
    //그룹을 합칠 때, 같은 그룹인지 체크해야 한다.
    int rootA = Find(A);
    int rootB = Find(B);
    //같은 그룹이라면 종료
    if (rootA == rootB) return;
    //대표 정보 업데이트
    parent[rootB] = rootA;
}
```

https://gist.github.com/hoconoco/e9f5f3761295c4ed6dc5b97ed45d1074

Union Find 의 경로 압축 1

Union Find로 구성된 집합 중 최악의 경우가 있다.

- 모든 노드가 일렬로 연결되었을 경우이다.
- Find() 를 할 때마다, N번의 연산이 일어나게 된다.
- Find(N) → N회 일어남



Union Find의 목표는 중간에 어떤 노드가 있는 지 궁금하지 않고,
그룹을 합치고, 어떤 그룹에 속해 있는 지가 중요하다.

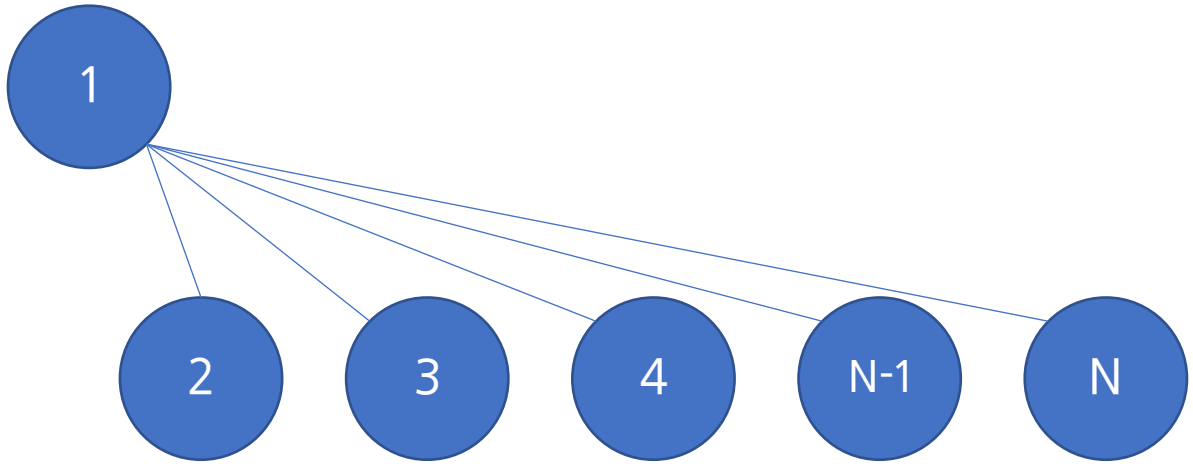
Union Find 의 경로 압축 2

경로 압축(pass compression)

root 노드와 다이렉트로 연결하기

1. **처음 조회할 경우, N번의 조회가 일어난다.**
2. **두번째부터는, 1회의 조회만 일어나도록 처리한다.**

Find() 업그레이드 필요



Find() 를 업데이트 한다.

- **왼쪽 코드에서 오른쪽 코드로 축약이 가능하다.**

```
//Find 특정 그룹의 대표 찾기 ( 재귀 )
int Find(int n) {
    //대표가 본인인 경우 return
    if (n == parent[n]) return n;
    //root 노드까지 찾아가기
    int root = Find( parent[n] );
    //root와 direct 연결
    parent[n] = root;
    //대표 반환
    return root;
}
```

```
//Find 특정 그룹의 대표 찾기 ( 재귀 )
int Find(int n) {
    //대표가 본인인 경우 return
    if (n == parent[n]) return n;
    //root 노드까지 찾아가기
    //해당 노드의 root 갱신
    //대표 반환 축약
    return parent[n] = Find(parent[n]);
}
```

https://gist.github.com/hoconoco/ed54cc9b489e93daebc8af1105e590f3

Union 한 그룹은 어떻게 쪼개나요?

Union한 결과를 쪼갤 수는 없다.

Union + Find 이므로, 합치기만 존재한다.

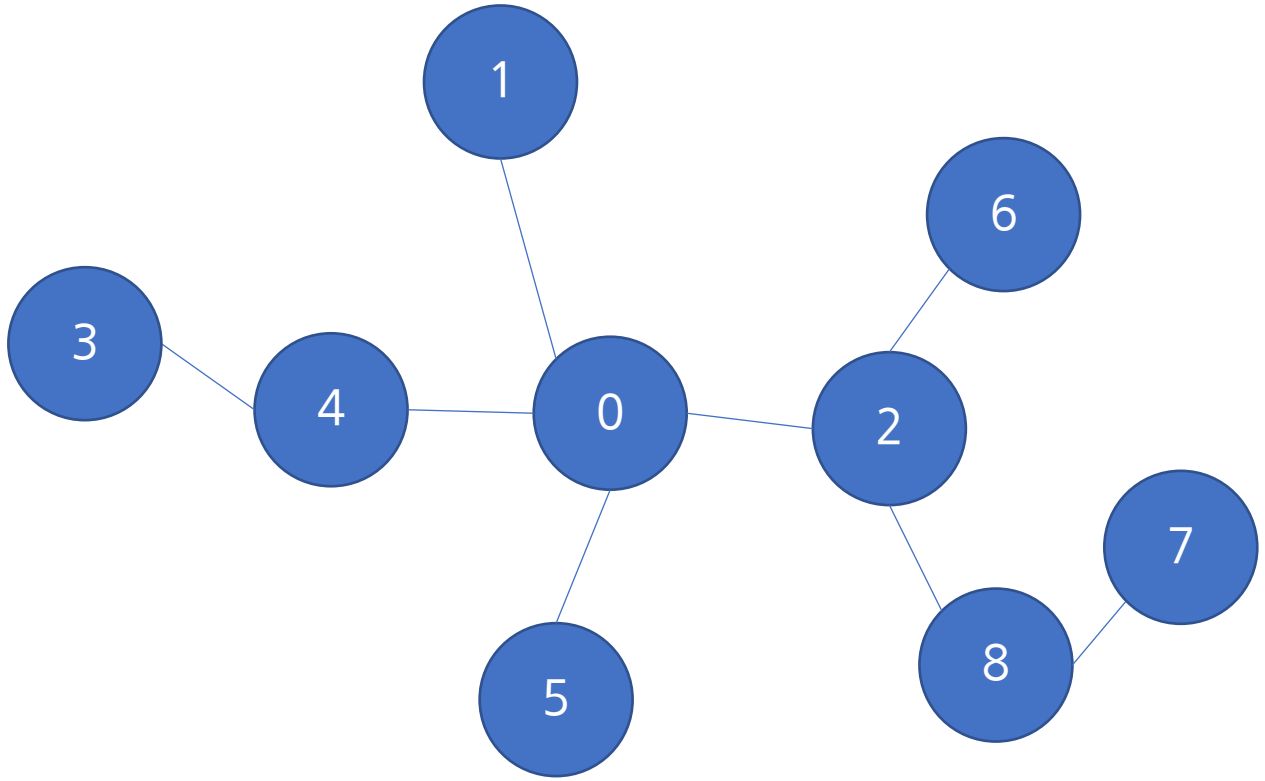
SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

MST with Union Find

최소 신장 트리

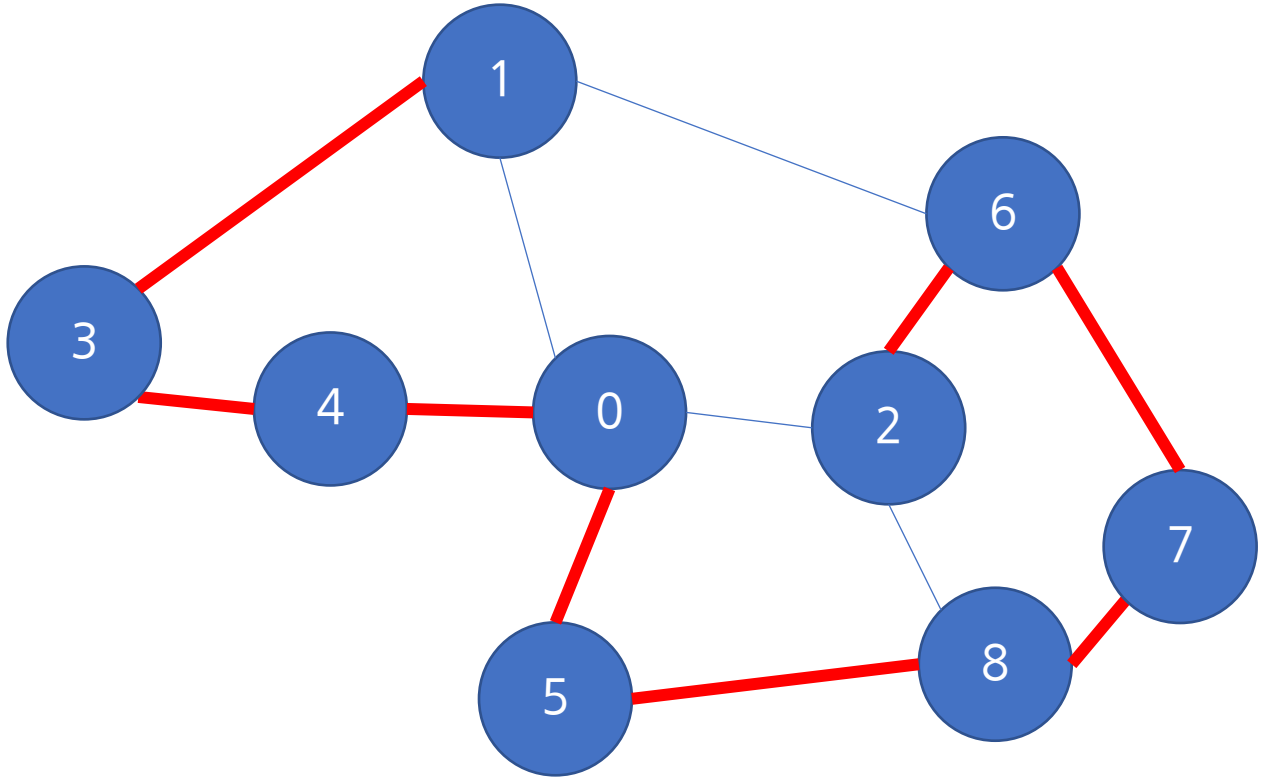
- Spanning Tree : 사이클 없이 모든 노드가 연결된 트리
- Minimum Spanning Tree : 가중치가 최소인 신장 트리



그래프의 부분 집합으로 존재

오른쪽과 같은 그래프가 있다.

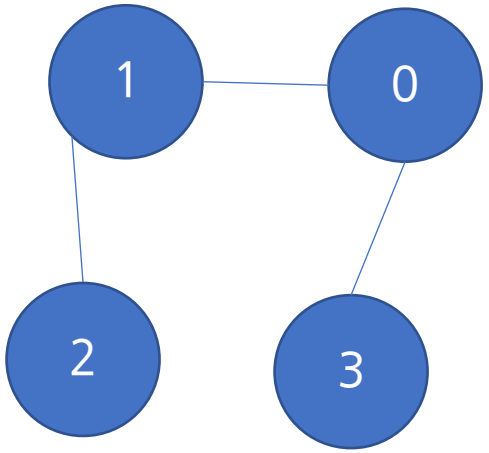
1. 그래프 내에 여러 개의 신장 트리가 있을 수 있다.
2. 여러 신장 트리 중 가중치가 최소인 MST가 존재한다.

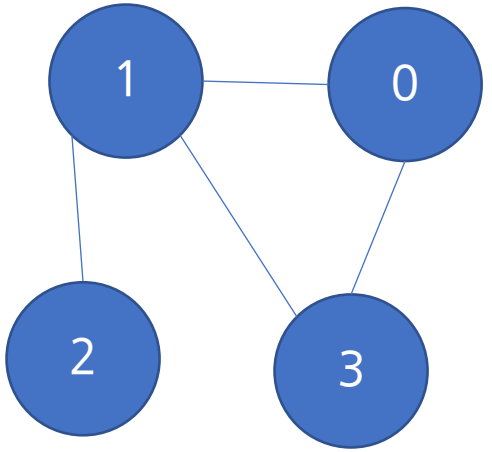


그래프 VS 트리

그래프와 트리는 여러 차이점이 있다.

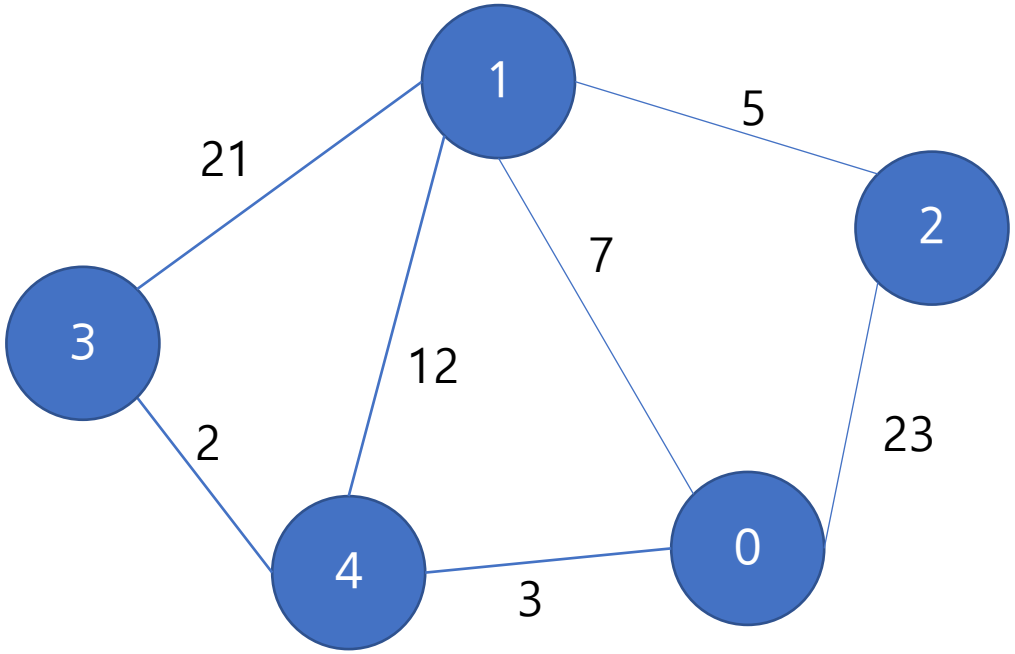
- 가장 큰 차이점은 **경로의 유일성**이다.
- 왼쪽은 트리, 오른쪽은 그래프이다.
- 0번 노드에서 2번 노드로 가고자 할 때, 트리는 1개의 경로, 그래프는 2개의 경로가 존재한다.





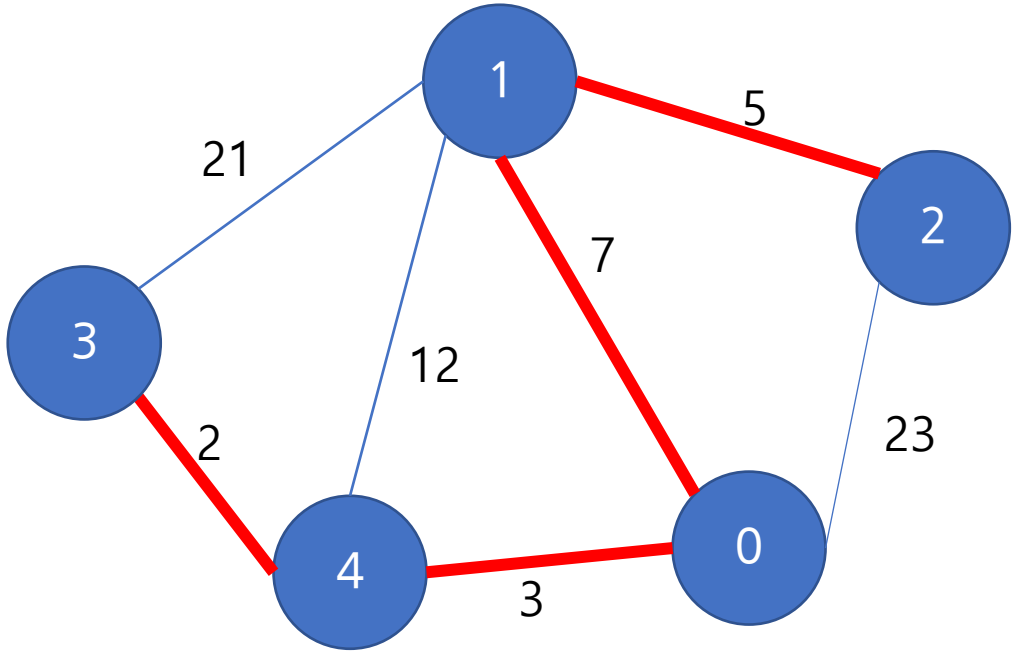
오른쪽과 같이 그래프 정보가 주어진다.

가중치가 최소인 MST를 찾아보자!



3 → 4 → 0 → 1 → 2 선택

- **2 + 3 + 7 + 5 = 17**



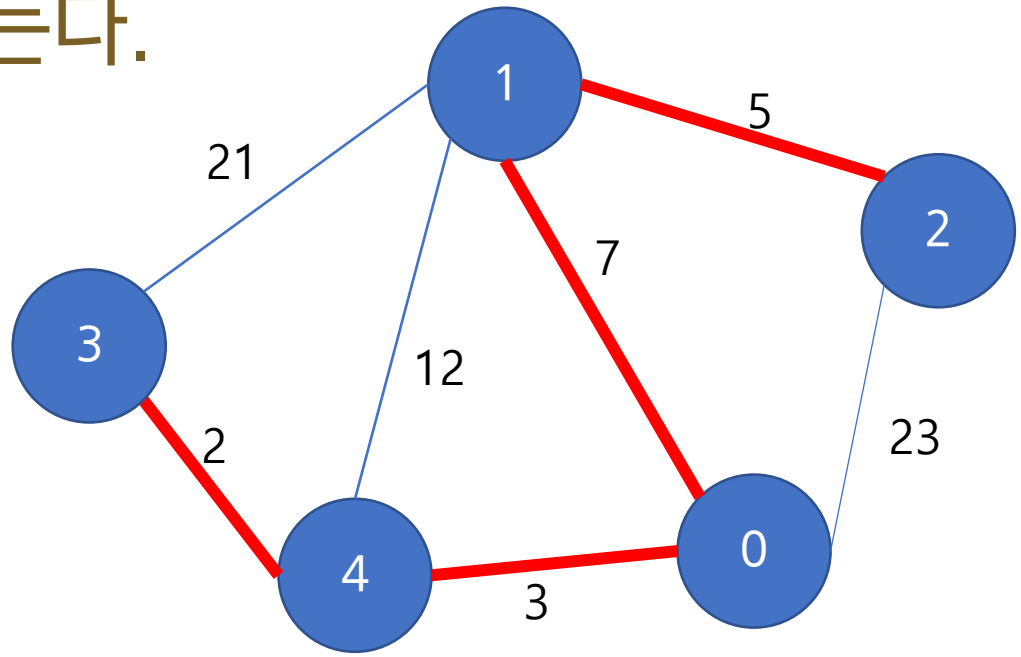
MST를 구하는 방법

MST 찾는 방법은 다음과 같다.

- 점(노드) 기준으로 찾기 : Priority Queue 사용 (추후에 학습할 예정)
- 선(Edge) 기준으로 찾기 : Union Find

간선의 가중치가 최소인 선 기준으로 찾는다.

- 3 - 4 : 2
- 4 - 0 : 3
- 1 - 2 : 5
- 1 - 0 : 7



코드로 구현하기

1. 데이터를 저장한다.

- 구조체를 만들어서 관리한다. (start, end, weight)
- start : 출발 노드 (본인 노드 번호)
- end : 도착 노드 (연결된 노드 번호)
- weight : 가중치 (정렬을 이용하므로 연산자 오버로딩 필요)

```
struct NODE {
    //출발 노드, 도착 노드, 가중치
    int start;
    int end;
    int weight;
};
```

2. Edge 정렬

- sort( )

```
//2. weight 기준 오름차순 정렬
sort(mst, mst + M);
```

3. 하나씩 연결한다.

- Union-Find 사용해서 같은 그룹인지 판별
- 같은 그룹이면 연결하지 않는다.

코드를 작성하고 빌드한다.

1. 데이터를 저장한다.

- 구조체를 만들어서 관리한다. (start, end, weight)
- start : 출발 노드 (본인 노드 번호)
- end : 도착 노드 (연결된 노드 번호)
- weight : 가중치 (정렬을 이용하므로 연산자 오버로딩 필요)

2. Edge 정렬

- sort( )

```
struct NODE mst[10];

int N, M;
cin >> N >> M;
//1. 데이터 저장
for (int i = 0; i < M; i++) {
    int s, e, w;
    cin >> s >> e >> w;
    mst[i] = { s,e,w };
}

//2. weight 기준 오름차순 정렬
sort(mst, mst + M);
```

https://gist.github.com/hoconoco/2f34c2fdfc0398d8c8d94df55353b8a6

3. 하나씩 연결한다.

- Union-Find 사용해서 같은 그룹인지 판별
- 같은 그룹이면 연결하지 않는다.

```
5 7
0 1 7
0 2 23
0 4 3
1 2 5
1 3 21
1 4 12
3 4 2
17
```

```
// parent 세팅 ( Union-Find )
for (int i = 1; i <= M; i++) {
    //본인이 대표
    parent[i] = i;
}

//MST의 최소값 저장할 변수
int sum = 0;
//3. 하나씩 연결하기
//맨 앞에 간선의 가중치가 최소부터 시작
for (int i = 0; i < M; i++) {
    struct NODE now = mst[i];
    //같은 그룹인 경우 무시
    if (Find(now.start) == Find(now.end)) continue;
    //다른 그룹이면 합치기
    Union(now.start, now.end);
    //가중치 누적
    sum += now.weight;
}
```

https://gist.github.com/hoconoco/11022e4190ce0d763a5fb8e5f394279e

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

내일 방송에서 만나요!

삼성 청년 SW 아카데미