유니온 파인드(Union Find)

유니온 파인드 (Union Find)

- 상호 배타적 집합(Disjoint-set) 이라고도 한다.
- 2가지 연산으로 이루어져 있다.
  ① Find : x가 어떤 집합에 포함되어 있는지 찾는 연산
  ② Union : x와 y가 포함되어 있는 집합을 합치는 연산
- 간단한 트리 구조를 이용해서 구현하며 parent[i] = i이면 root를 나타낸다.
- Union(x, y) 연산 : Parent[y] = x → y의 부모가 x가 된다 → y는 x와 같은 집합이다.
- Find(x) : x의 root를 반환한다.

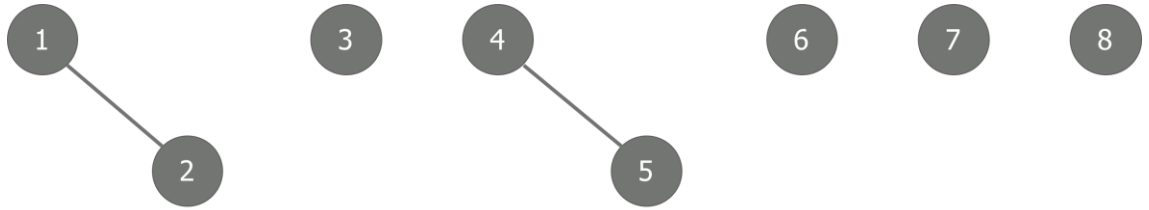
i	1	2	3	4	5	6	7	8
P[i]	1	2	3	4	5	6	7	8

유니온 파인드 (Union Find)

Union(1,2)

i	1	2	3	4	5	6	7	8
P[i]	1	1	3	4	5	6	7	8

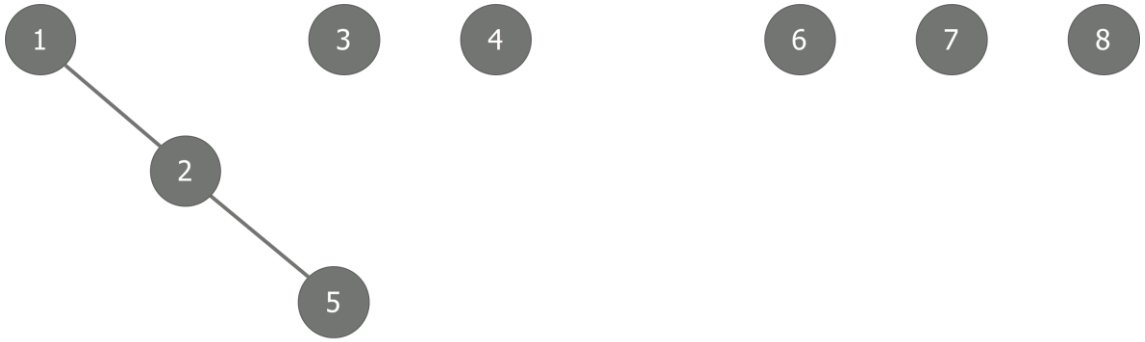
Union(4,5)



i	1	2	3	4	5	6	7	8
P[i]	1	1	3	4	4	6	7	8

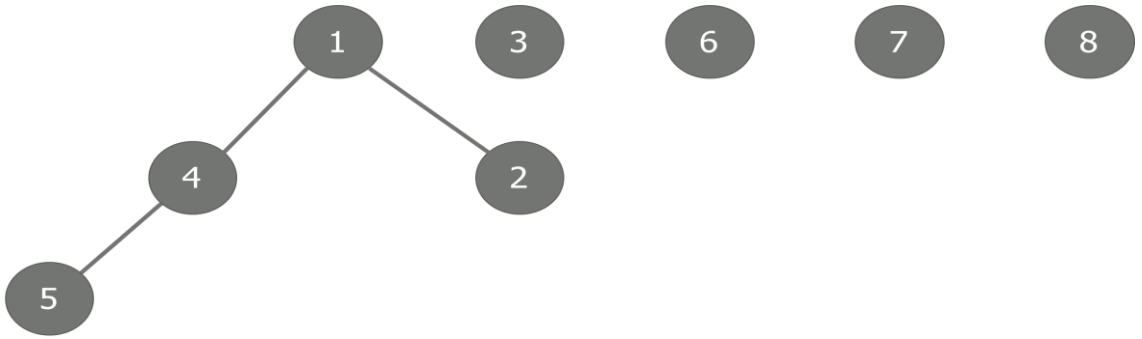
유니온 파인드 (Union Find)

Union(2,5) => 바로 Union만 하면 문제 발생!! 4와 5가 집합이었으나 지금은 집합이 아님



i	1	2	3	4	5	6	7	8
P[i]	1	1	3	4	2	6	7	8

Union(2, 5)를 하기 위해서 Find(2)와 Find(5)를 통해 root를 접근 한 후 루트를 Union함
→ Union(Find(2), Find(5)) → Union(1, 4)



i	1	2	3	4	5	6	7	8
P[i]	1	1	3	1	4	6	7	8

Gaussian

유니온 파인드 (Union Find)

- Find의 재귀 호출 구현

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

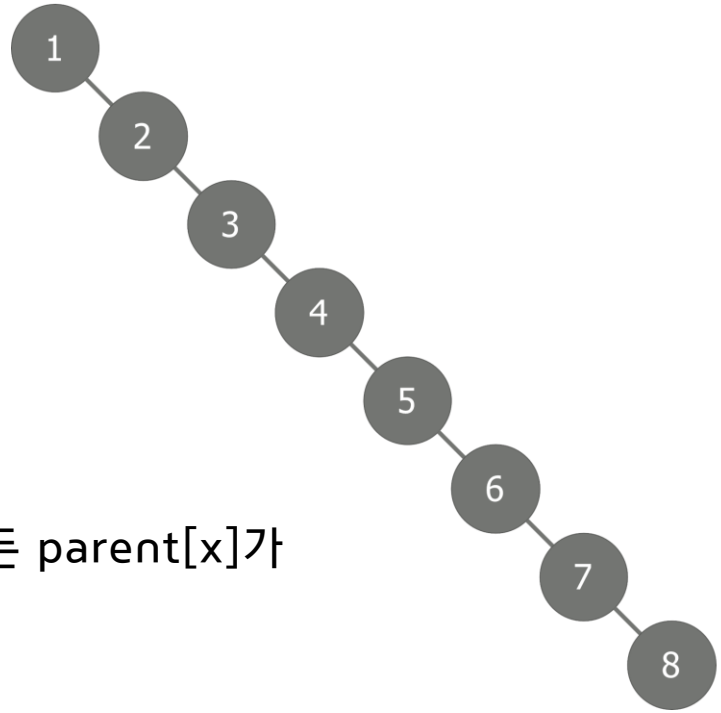
- Union의 구현 : Union(x, y) → y의 parent를 x의 parent로 설정한다.

```
int Union(int x, int y) {
    x = Find(x);
    y = Find(y);
    parent[y] = x;
}
```

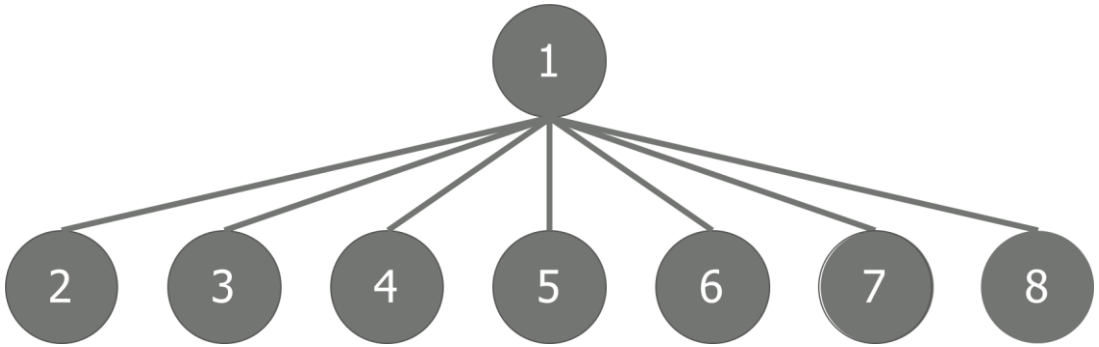
유니온 파인드 (Union Find)

- Union의 구현
- Union(x, y) → y의 parent를 x로 설정한다.

```
int Union(int x, int y) {
    x = Find(x);
    y = Find(y);
    parent[y] = x;
}
```

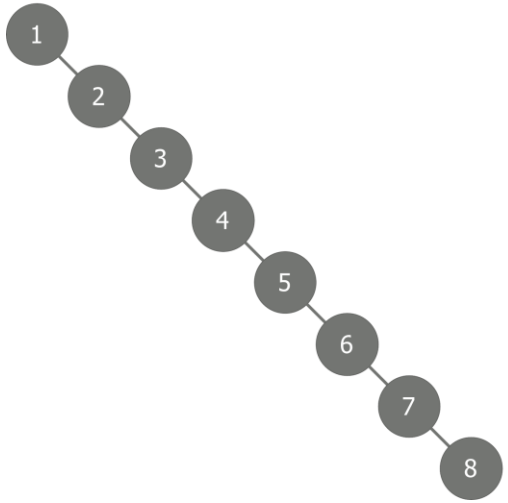


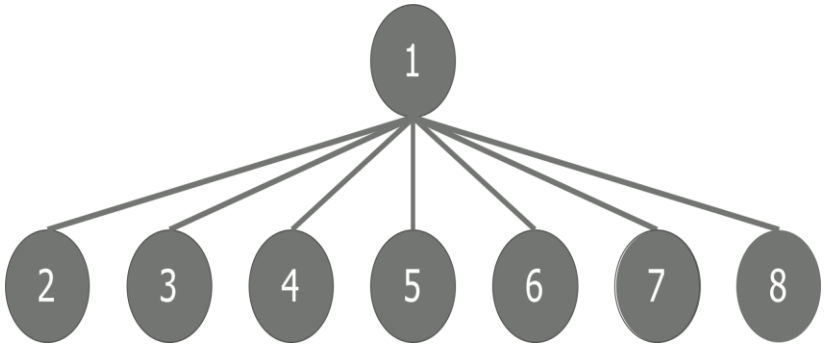
- 오른쪽 그림과 같은 문제가 생길 수 있다.
- 이렇게 되면 find의 시간 복잡도가 O(N)이 된다.
  - ☞ Find(x)의 값은 최종적인 root를 반환하므로 모든 parent[x]가 최종적인 root값만 저장하고 있으면 된다.



유니온 파인드 (Union Find)

- Find(8)을 할 때, 1까지 올라가면서 만난 값은 모두 8과 같은 집합에 포함되어 있다.
- Find를 하면서 만난 모든 값의 parent를 루트로 바꾸어 주면 아래와 같이 변하게 된다.
  ☞ 경로 압축






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

```
int Find(int x) {
    if (x == parent[x]) {
        return x;
    } else {
        int y = Find(parent[x]);
        parent[x] = y;
        return y;
    }
}
```

Gaussian

유니온 파인드 (Union Find)

```
int Find(int x) {
    if (x == parent[x]) {
        return x;
    } else {
        int y = Find(parent[x]);
        parent[x] = y;
        return y;
    }
}
```

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

Gaussian