


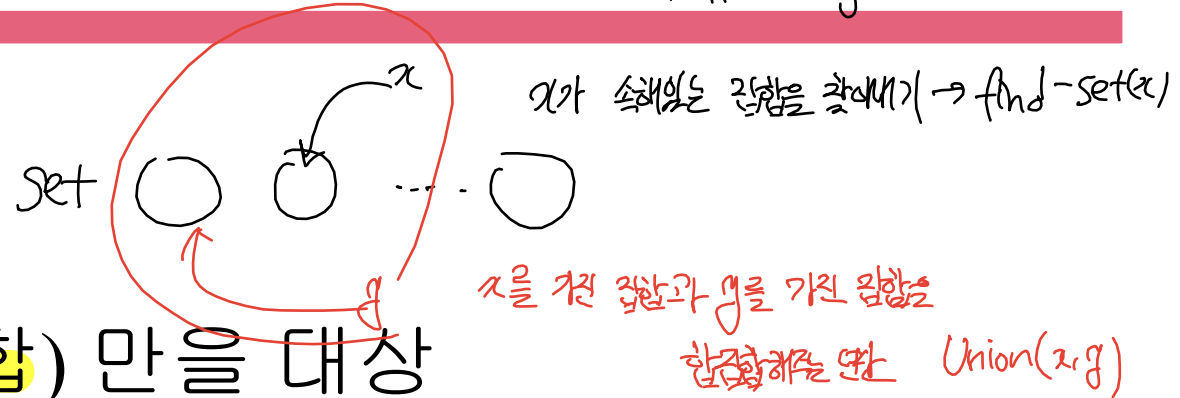
# 상호 배타적 집합의 처리



*mutual disjoint set*

# 집합(Set)의 처리

각은 여러개X disjoint set 일.



- disjoint set(배타적 집합) 만을 대상

- 교집합은 공집합

- 연산

각 원소 한개씩을 가진 집합을 만들어주는 제일 기본적인 연산

- Make-Set( $x$ ): 원소  $x$ 로만 이루어진 집합을 만든다
- Find-Set( $x$ ): 원소  $x$ 를 가지고 있는 집합을 알아낸다
- Union( $x, y$ ): 원소  $x$ 를 가진 집합과 원소  $y$ 를 가진 집합의 합집합

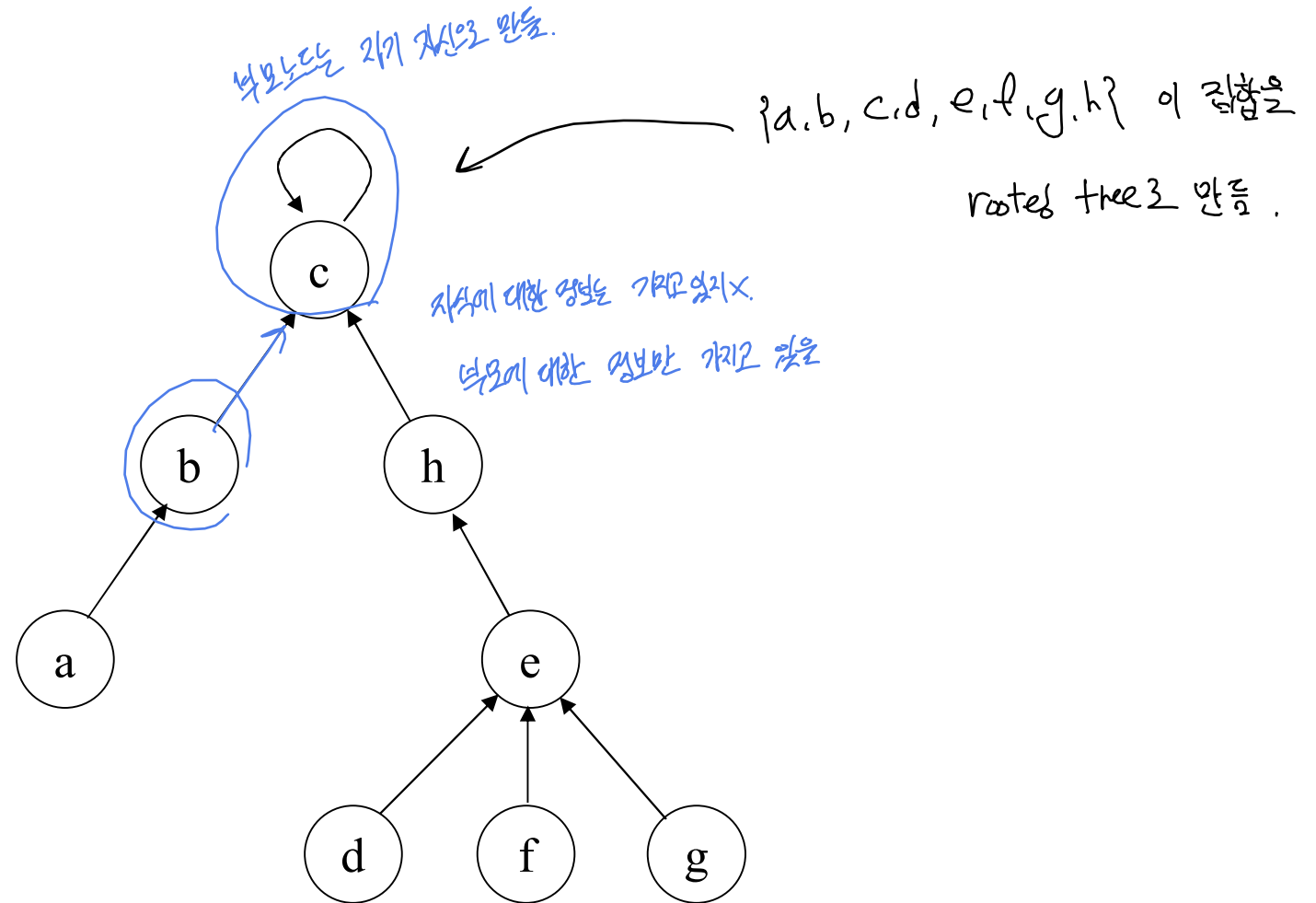
- tree를 이용하는 방법

# Tree를 이용한 처리

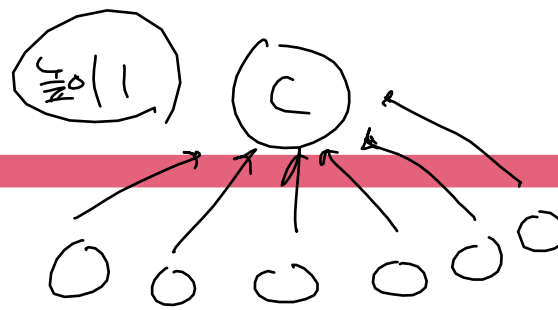
→ Rooted tree 집합을 대표하는 원소

- 같은 집합의 원소들은 하나의 tree로 관리한다
  - child가 parent를 가리킨다
- tree의 root를 집합의 대표 원소로 삼는다

# Tree를 이용한 집합 표현의 예



# 두 집합의 합집합

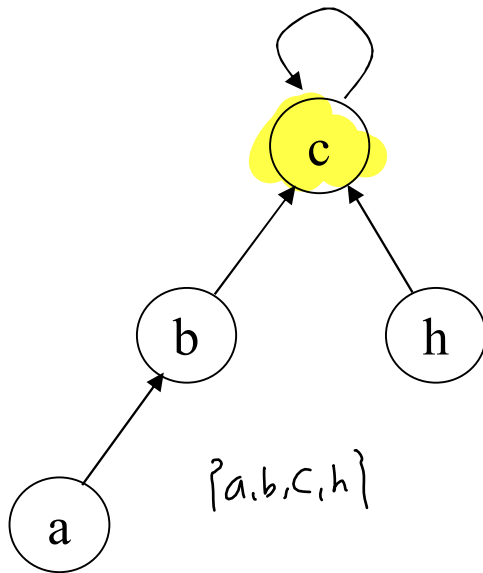


뒤의 parent를 올라 가는데  
시간이 걸림

최대 높이 만큼 걸림

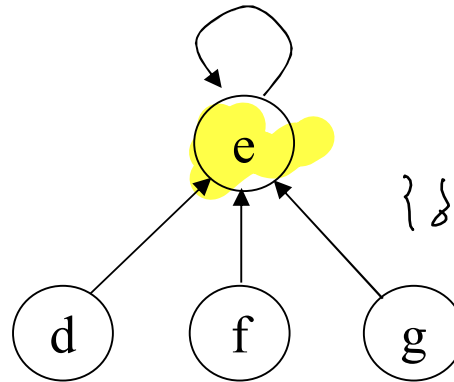
$O(h)$

Union 연산



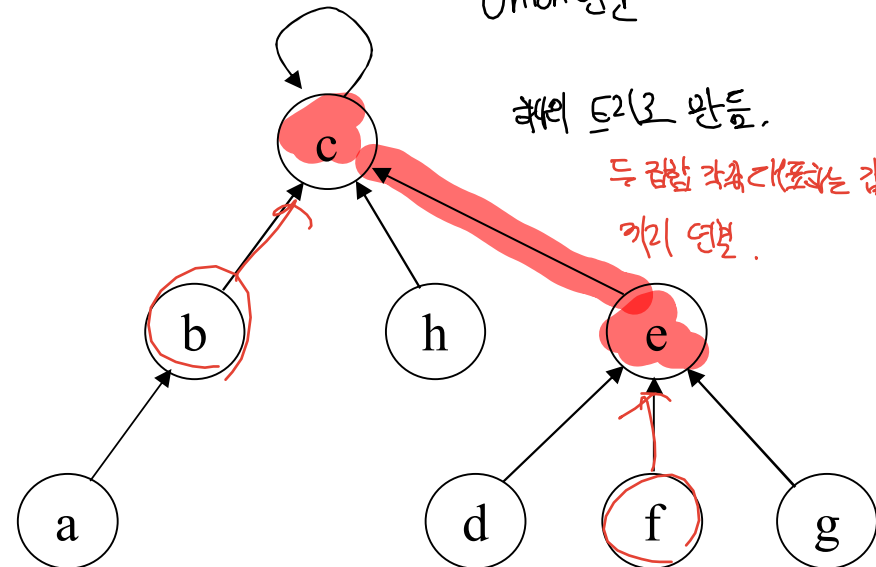
$\{a, b, c, h\}$

+



$\{d, e, f, g\}$

=



해의 트리로 만들.

두 집합 각 요소에 해당하는 값  
끼리 연결.

$find-set(b)$

$find(f)$

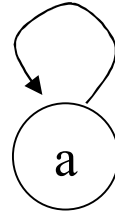
2번

$f \rightarrow e \rightarrow c$

root를 만날 때까지 찾아감

# 하나의 원소로 이루어진 집합

---



# Tree를 이용한 집합 처리 알고리즘

Algorithm Make-Set( $x$ )      ▷ 노드  $x$ 를 유일한 원소로 하는 집합을 만든다.

$$\{ p[x] \leftarrow x \} x$$

Algorithm Union( $x, y$ )      ▷ 노드  $x$ 가 속한 집합과 노드  $y$ 가 속한 집합을 합친다

```

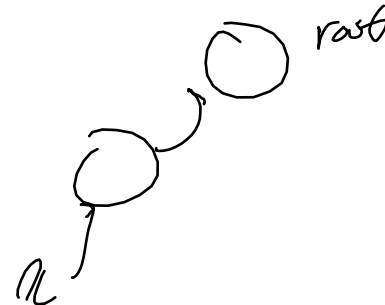
{
    p[Find-Set(y)] ← Find-Set(x);    root를 바꿔주는 일만
}

```

**Algorithm Find-Set( $x$ )**

- ▷ 노드  $x$ 가 속한 집합을 알아낸다.
- 노드  $x$ 가 속한 트리의 루트 노드를 반환한다.

```
{
    if ( $x = p[x]$ )
        then return  $x$ 
        else return Find-Set( $p[x]$ )
}
```



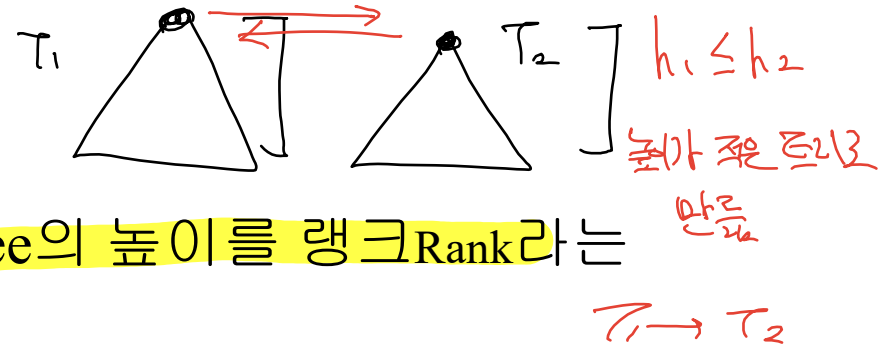
α의 level과 영한시간과  
β의 양.

# 연산의 효율을 높이는 방법

$T_1 \cup T_2$

$T_1$  루트에 연결할 것인가

$T_2$  루트에 연결할 것인가



- Rank를 이용한 Union

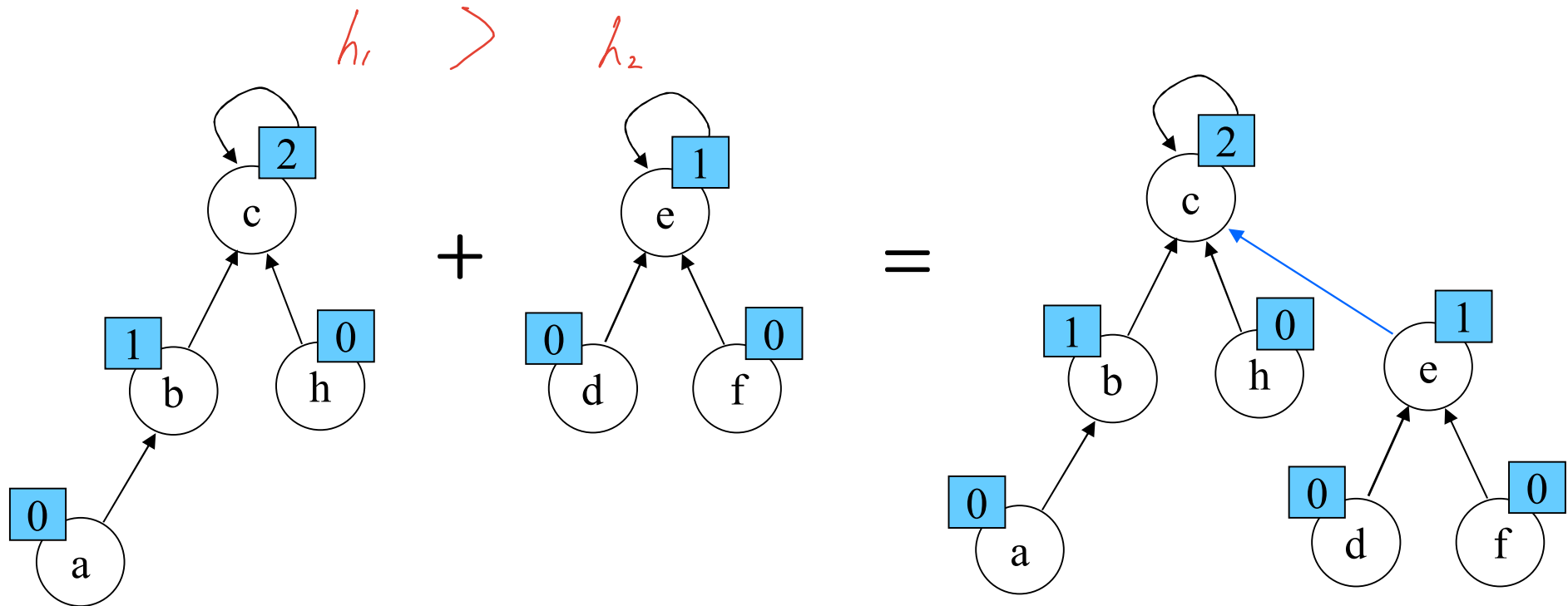
- 각 노드는 자신을 루트로 하는 subtree의 높이를 랭크 Rank라는 이름으로 저장한다
- 두 집합을 합칠 때 rank가 낮은 집합을 rank가 높은 집합에 붙인다

- Path compression

- Find-Set을 행하는 과정에서 만나는 모든 노드들이 직접 root를 가리키도록 포인터를 바꾸어 준다

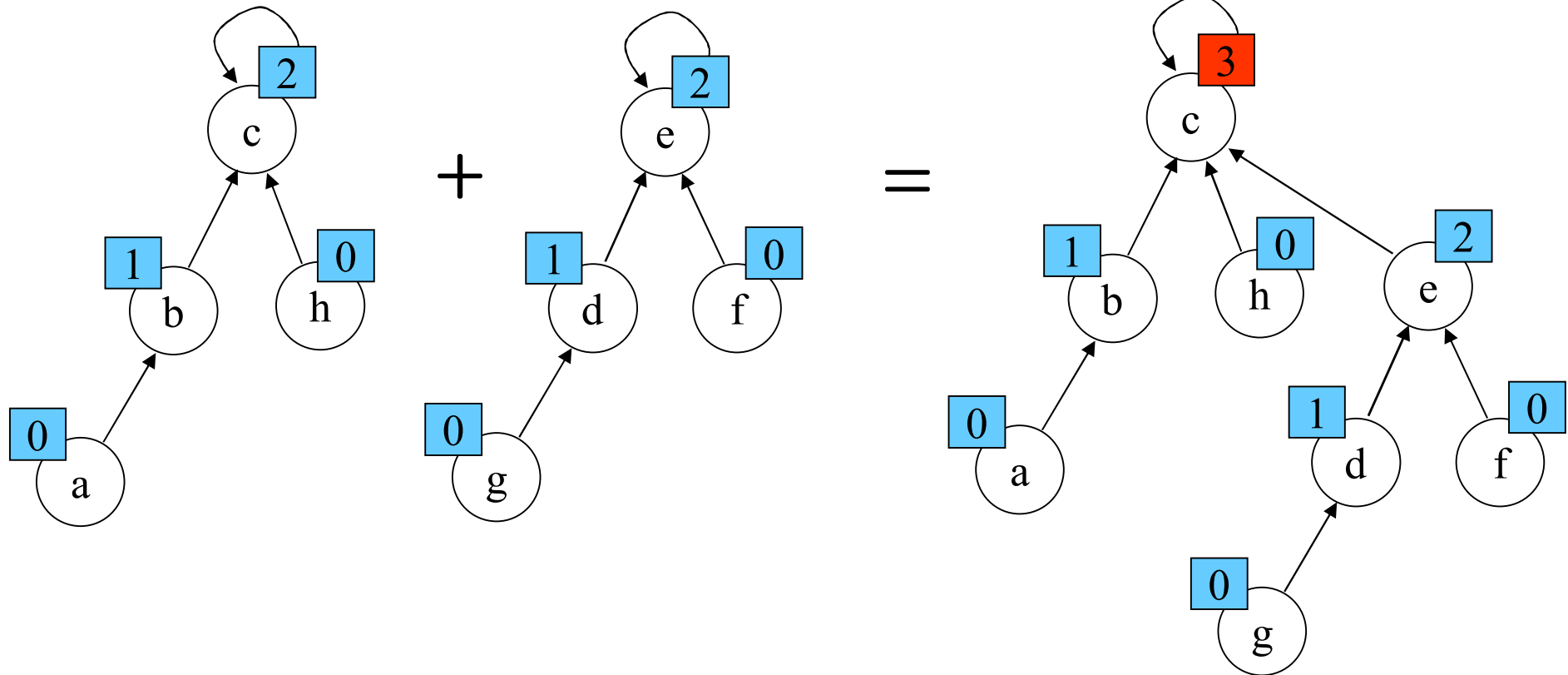


# 랭크를 이용한 Union의 예

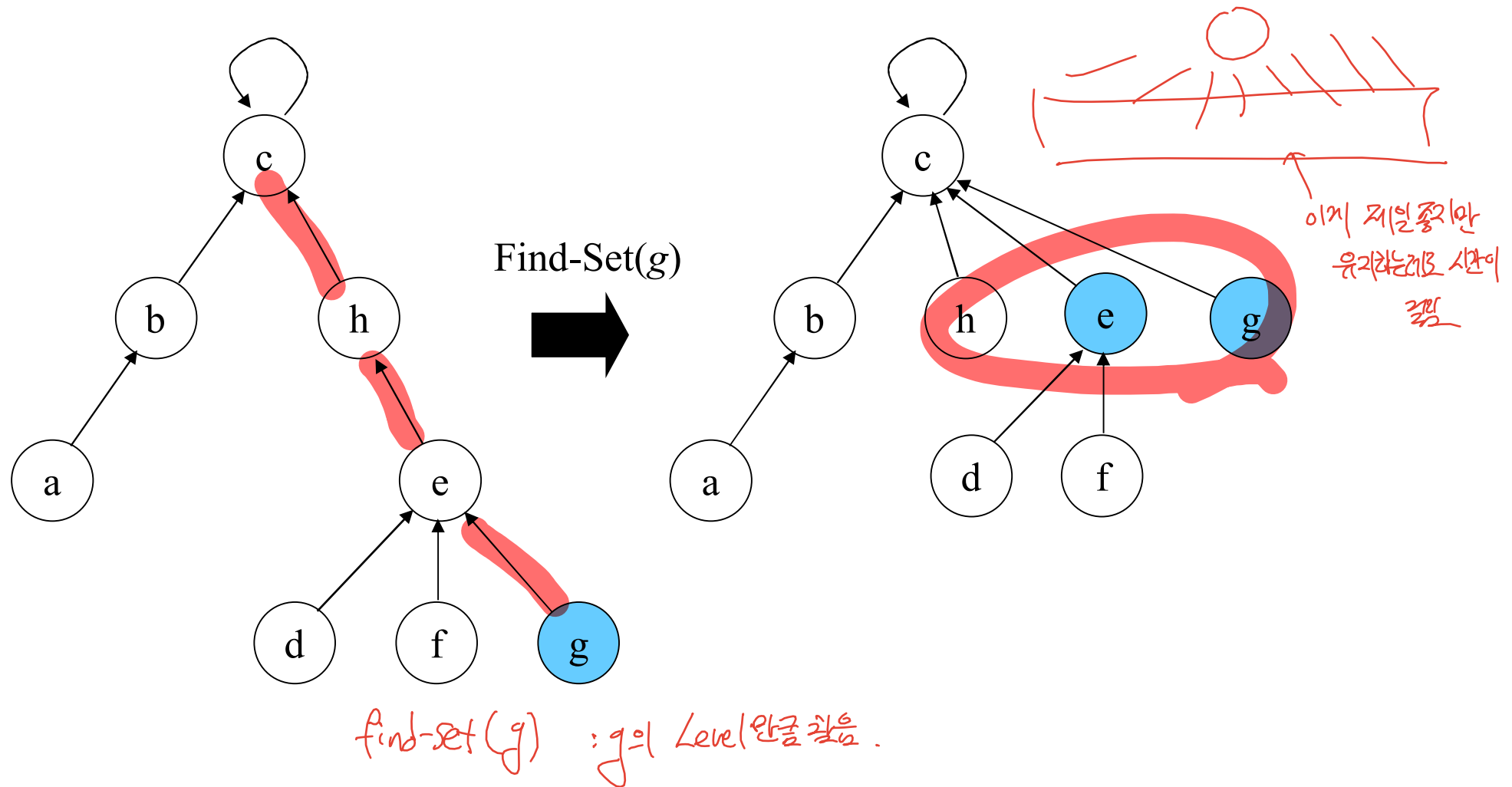


# 랭크를 이용한 Union에서 랭크가 증가하는 예

높이가 같은 경우



# Path Compression의 예



# Rank를 이용한 Union과 Make-Set

Algorithm Make-Set( $x$ )

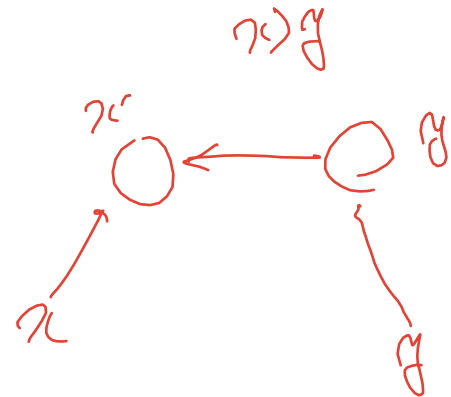
$p[x] \leftarrow x$   
 $\text{rank}[x] \leftarrow 0$

Algorithm Union( $x, y$ )

$x' \leftarrow \text{Find-Set}(x)$   
 $y' \leftarrow \text{Find-Set}(y)$   
**if** ( $\text{rank}[x'] > \text{rank}[y']$ )  
    **then**  $p[y'] \leftarrow x'$   
    **else**

$p[x'] \leftarrow y'$

**if** ( $\text{rank}[x'] = \text{rank}[y']$ ) **then**  $\text{rank}[y'] \leftarrow \text{rank}[y'] + 1$



같은 경우

# Path Compression을 이용한 Find-Set

Algorithm Find-Set( $x$ )

```
if ( $p[x] \neq x$ ) then  $p[x] \leftarrow \text{Find-Set}(p[x])$   
return  $p[x]$ 
```

# 수행시간

[정리]

Tree를 이용해 표현되는 Disjoint set에서 랭크를  
이용한 **Union**과 **경로압축을 이용한 Find-Set**을  
동시에 사용하면, m번의 Make-Set, Union, Find-  
Set 중 n번이 ~~Make-Set~~일 때 이들의 수행시간은  
 $O(m \log^* n)$ 이다. *Find-set*

*원소 m개*

$O(m)$

*Union, find-set 99% 많음*

$O(m)$

*상한인가에요 -*

$$\log^* n = \min \{k : \underbrace{\log \log \dots \log n}_k \leq 1\}$$

사실상 linear time임

# Set Union/Find 응용

Kruskal 알고리즘

// 입력: 에지에 가중치가 있는 그래프  $G = (V, E)$

$R = E$ ; //  $R$ 은 남아 있는 에지들의 집합

$F = \phi$ ; // 현재까지 구성한 포리스트 에지들의 집합

while ( $|F| \neq n-1$ )

$R$ 에서 최소 가중치의 에지  $(v, w)$ 를 제거한다.

if ( $(v, w)$ 를  $F$ 에 추가할 때 사이클을 만들지 않으면)

$(v, w)$ 를  $F$ 에 추가한다

$(v, w)$ 를  $F$ 에 추가할 때 사이클이 만들어지는지  
check하는데 이용

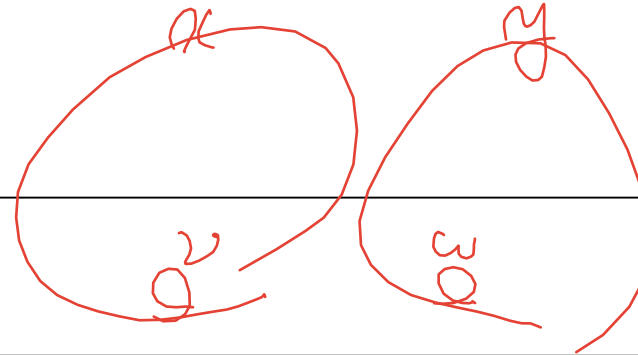
Union find 이용

# Set Union/Find 응용 – Kruskal 알고리즘

```
// set 초기화
```

```
모든 정점  $u$ 에 대하여,
```

```
Make-Set( $x$ )
```



```
// ( $v, w$ )를  $F$ 에 추가할 때 사이클이 만들어지는지 검사
```

```
 $x = \text{Find-Set}(v)$ 
```

```
 $y = \text{Find-Set}(w)$ 
```

```
if ( $x \neq y$ ) // 사이클이 만들어지지 않으면 다른 세트로
```

```
// ( $v, w$ )를  $F$ 에 추가
```

```
//  $v \Rightarrow w$ 가 속해있는 트리와  $w$ 가 속해있는 트리를 연결하여 하나의 트리로 만듦
```

```
Union( $x, y$ )
```

하나라의 subtree에 있는 노드들은  
하나라 집합임.

else

( $v, w$ ) 무시



# Kruskal 알고리즘 (욕심쟁이 알고리즘)

```
// 입력: 에지에 가중치가 있는 그래프  $G = (V, E)$   
R = E;    // R은 남아 있는 에지들의 집합  
F =  $\phi$ ;   // 현재까지 구성한 포리스트 에지들의 집합  
while ( $|F| \neq n-1$ )  
    R에서 최소 가중치의 에지  $(v,w)$ 를 제거한다.  
    if ( $(v,w)$ 를 F에 추가할 때 사이클을 만들지 않으면)  
         $(v,w)$ 를 F에 추가한다;
```

Kruskal 알고리즘 수행시간:

m: 에지 수

에지들을 정렬하는데 걸리는 시간  $O(m \log m)$

while loop: 거의  $O(m)$

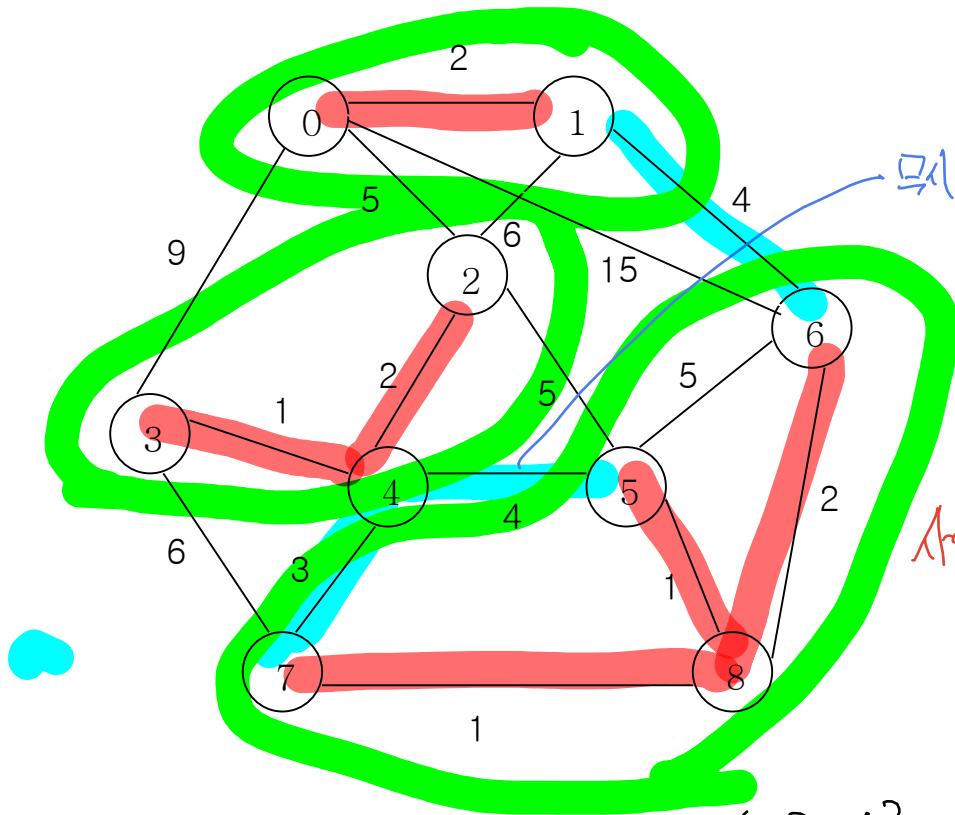
=> 전체 수행시간은  $O(m \log m)$

edge 개수 m

$\approx O(m)$

# Kruskal 알고리즘 예

## ● 예



$\{0,1\} \{2\} \{3\} \{4\} \dots \{8\}$

중복되지 않은 간선만 포함.

$(1,8) \rightarrow$  추가  $\{1,8\}$   
 $\text{Find}(1) \text{Find}(8)$

$(5,8) \rightarrow$   $\{5,8\}$   
 $\text{Find}(5) \text{Find}(8)$

$(3,4)$   $\{5,7,8\}$   
 $\text{Find}(3) \text{Find}(4)$   $\{3,4\}$

$(0,1)$   $\text{Find}(0) \text{Find}(1)$   $\{0,1\}$

$\{0,1\} \{2,3,4\} \{5,6,7,8\}$  일일이 두 집합에 속하는 edge를 추가하면 cycle 만들어요.

$\{4,7\}$

$\{2,3,4,5,6,7,8\}$

$\rightarrow$  tree이 됨