



쉽게 배우는 알고리즘

7장. 상호 배타적 집합의 처리

7장. 상호 배타적 집합의 처리

얼마 전 나는 학술회의 준비차 다원을 읽었다.
읽으면서 그 동안 읽지 않기를 잘했다는 생각이 들었다.
다른 때 다원을 읽었더라면 여전히 이해할 수 없었을 것이기 때문이다.
... 결국, 읽을 준비가 되었을 때 읽어야 한다는 것이다.

-로저 생크

학습목표

- 연결 리스트를 이용한 상호배타적 집합의 처리 방법을 이해한다.
- 연결 리스트를 이용한 집합의 처리를 위한 연산들의 수행시간을 분석할 수 있도록 한다.
- 트리를 이용한 상호배타적 집합의 처리 방법을 이해한다.
- 트리를 이용한 집합의 처리를 위한 연산들의 수행시간을 기본적인 수준에서 분석할 수 있도록 한다.

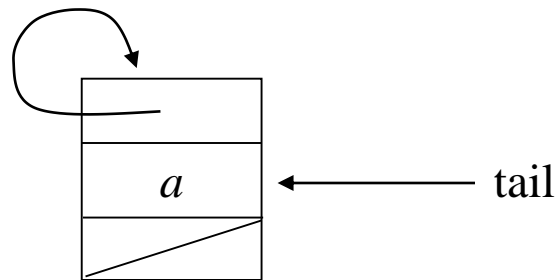
Set의 처리

- 이 장에서는 disjoint set(배타적 집합) 만을 대상으로 한다
- 그러므로 교집합은 없다
- 지원할 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
- Linked list를 이용하는 방법과 tree를 이용하는 방법이 있다

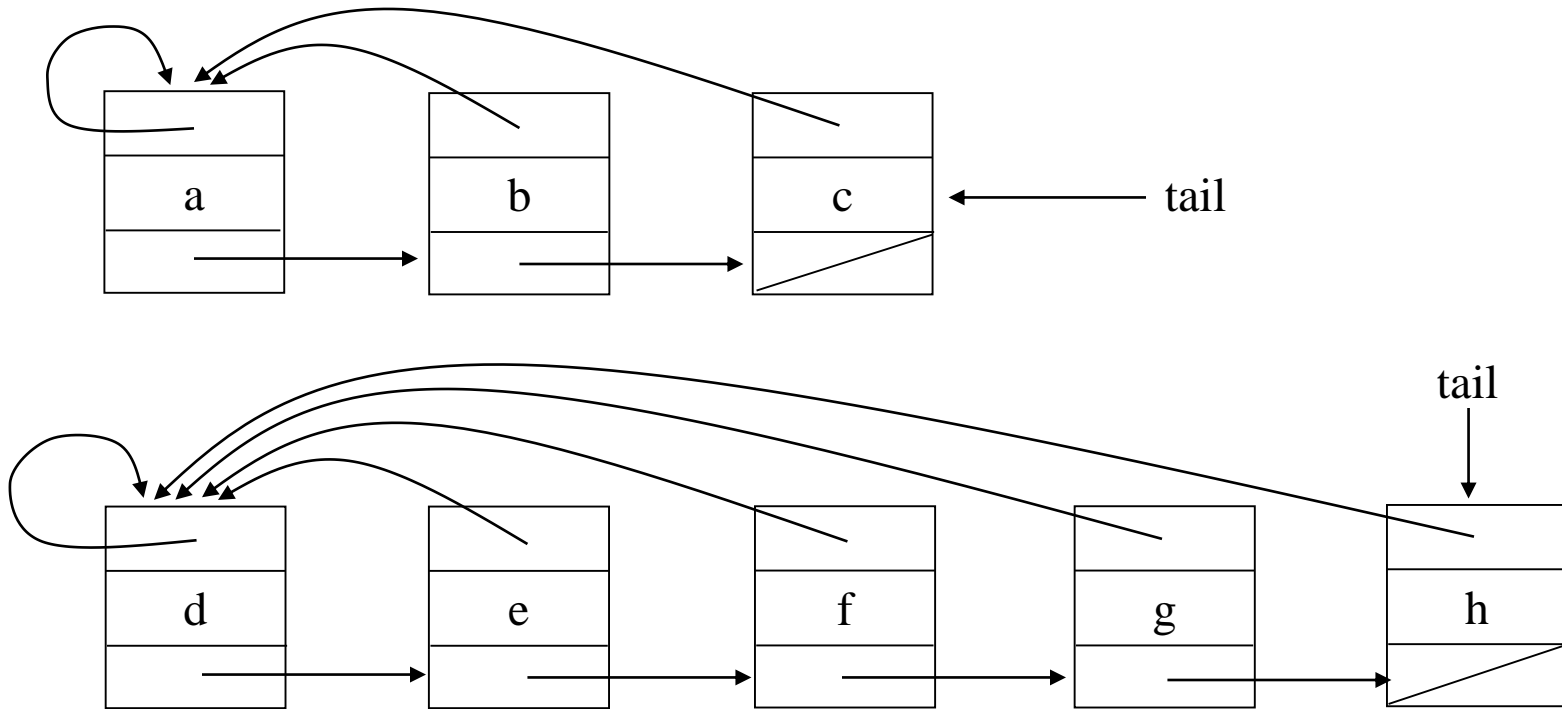
Linked List를 이용한 처리

- 같은 집합의 원소들은 하나의 linked list로 관리한다
- Linked list의 맨 앞의 원소를 집합의 대표 원소로 삼는다

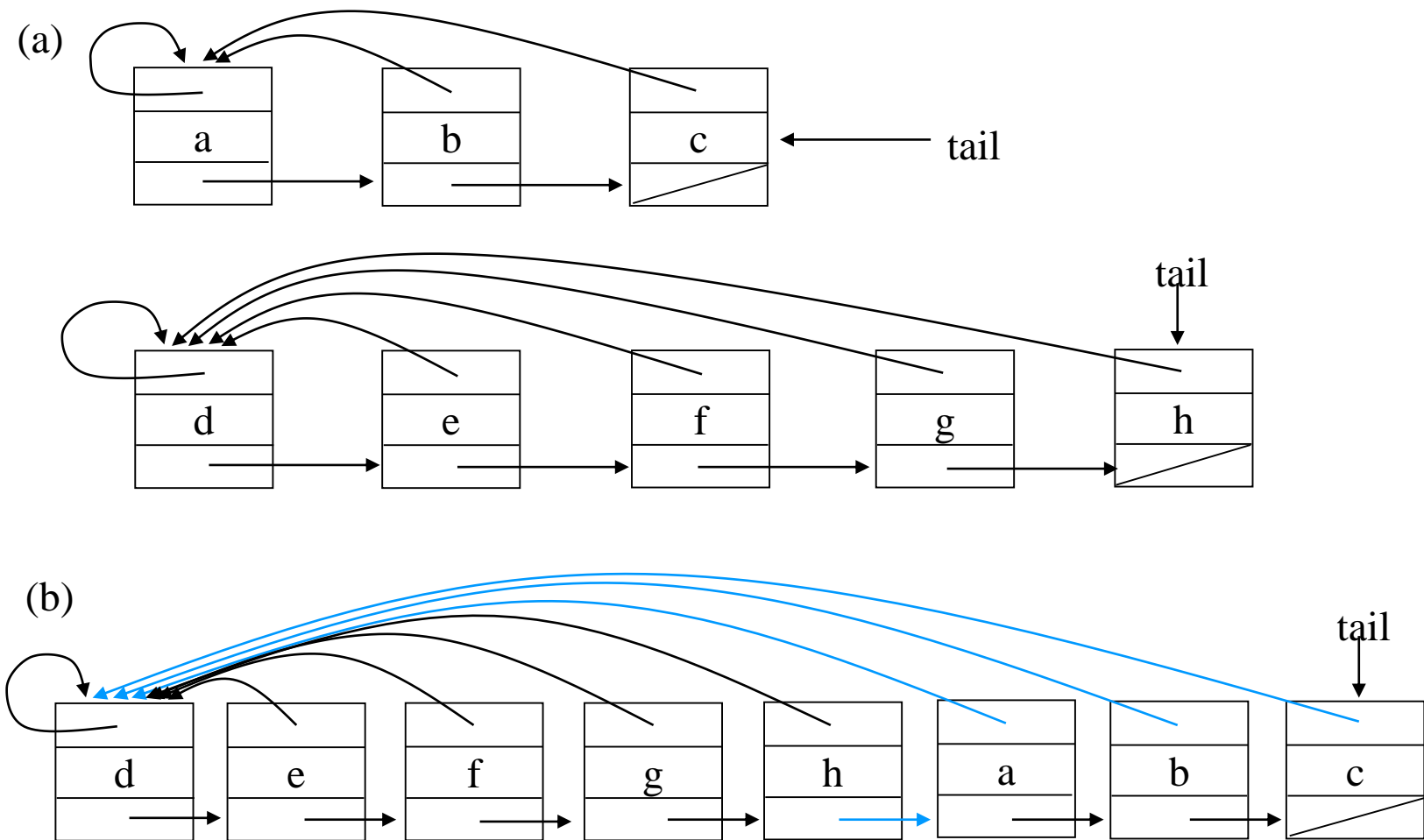
하나의 원소로 이루어진 집합



Linked List로 된 두 집합



합집합을 만드는 예



파란색은 변동이 생긴 간선들

Weight을 고려한 Union

- Linked list로 된 두 집합을 합할 때 작은 집합을 큰 집합의 뒤에 붙인다
 - 대표 원소를 가리키는 포인터 갱신 작업을 최소화하기 위한 것

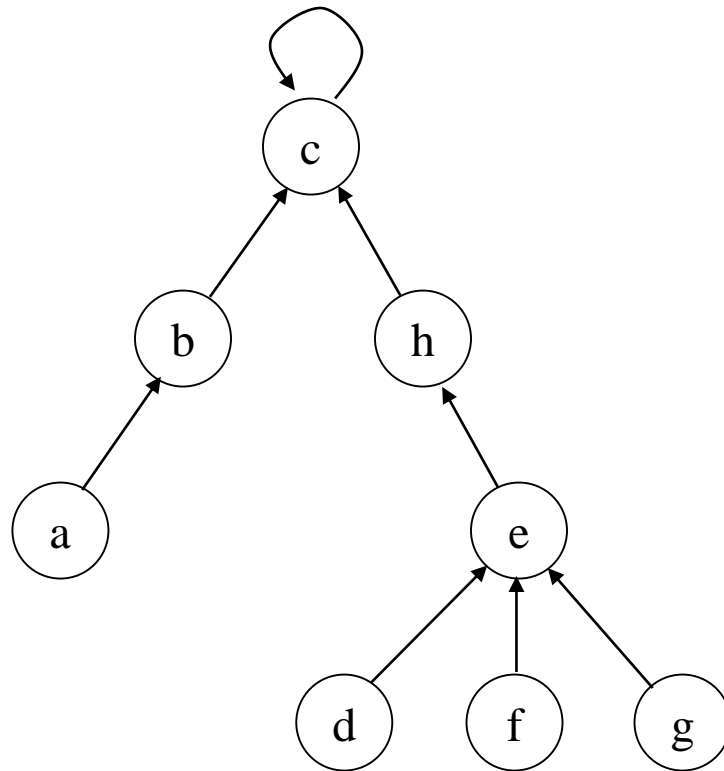
[정리 1]

Linked list를 이용해 표현되는 배타적 집합들을 만들면서 **Weight**을 고려한 **Union**을 사용할 때, m 번의 Make-Set, Union, Find-Set 중 n 번이 Make-Set이라면 이들의 총 수행시간은 $O(m + n \log n)$ 이다.

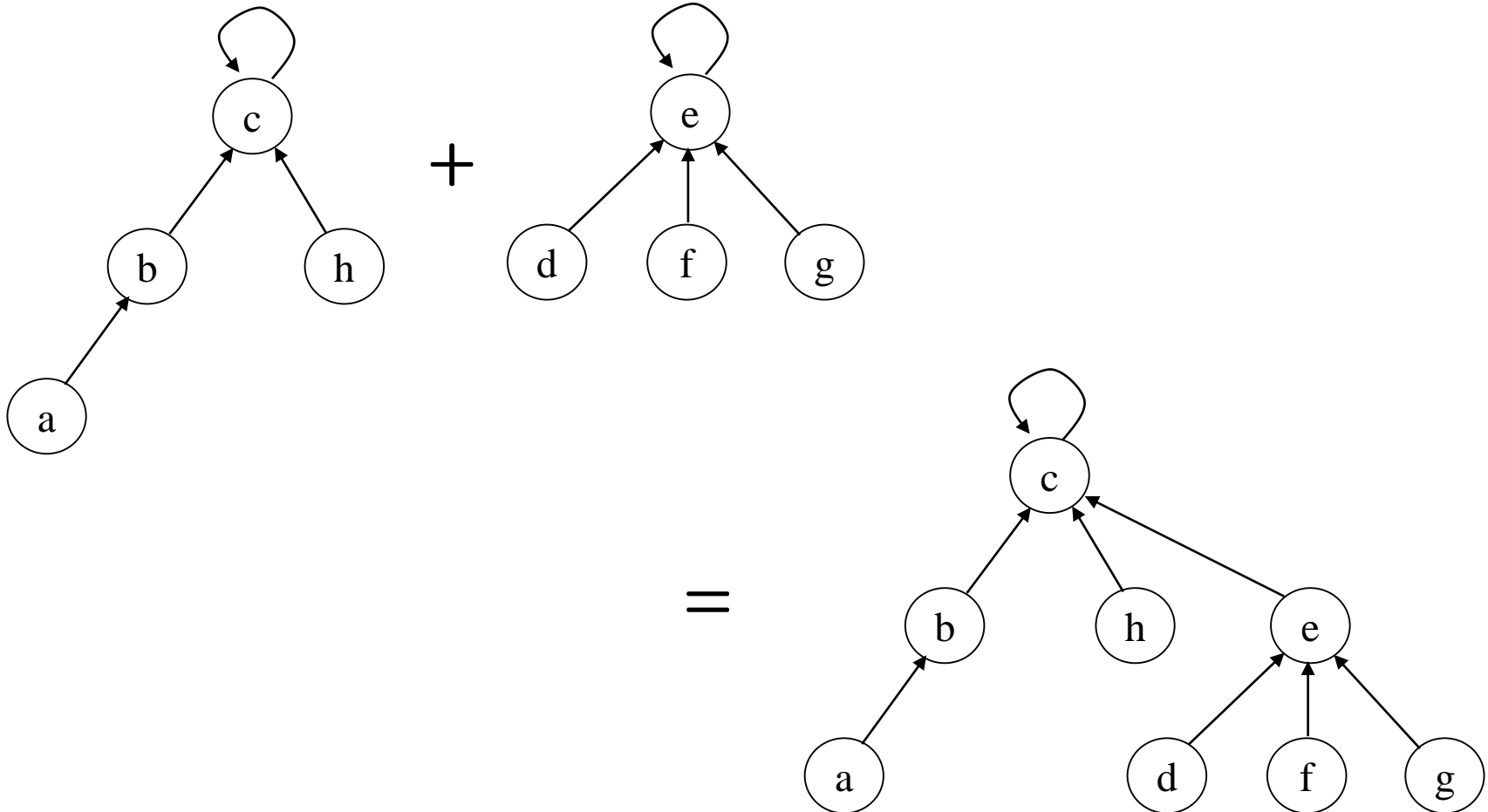
Tree를 이용한 처리

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

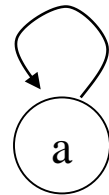
Tree를 이용한 집합 표현의 예



두 집합의 합집합



하나의 원소로 이루어진 집합



Tree를 이용한 집합 처리 알고리즘

Make-Set(x) ▷ 노드 x 를 유일한 원소로 하는 집합을 만든다.
{
 $p[x] \leftarrow x$;
}

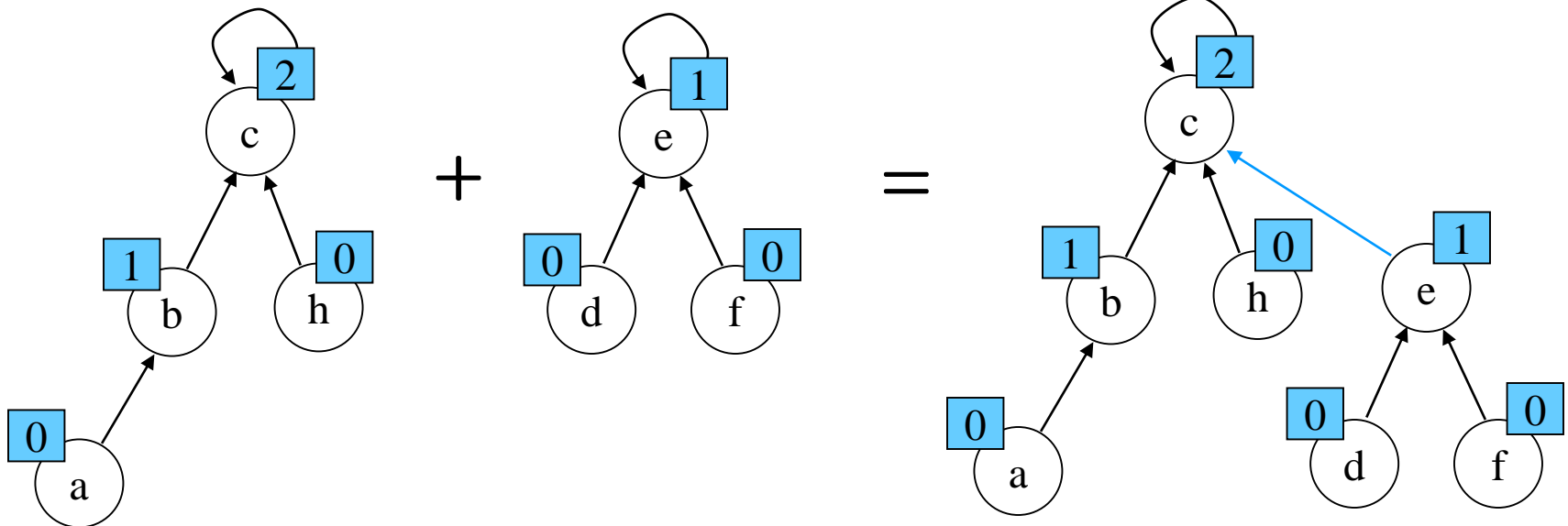
Union(x, y) ▷ 노드 x 가 속한 집합과 노드 y 가 속한 집합을 합친다
{
 $p[\text{Find-Set}(y)] \leftarrow \text{Find-Set}(x)$;
}

Find-Set(x) ▷ 노드 x 가 속한 집합을 알아낸다.
 노드 x 가 속한 트리의 루트 노드를 리턴한다.
{
 if ($x = p[x]$)
 then return x ;
 else return Find-Set($p[x]$) ;
}

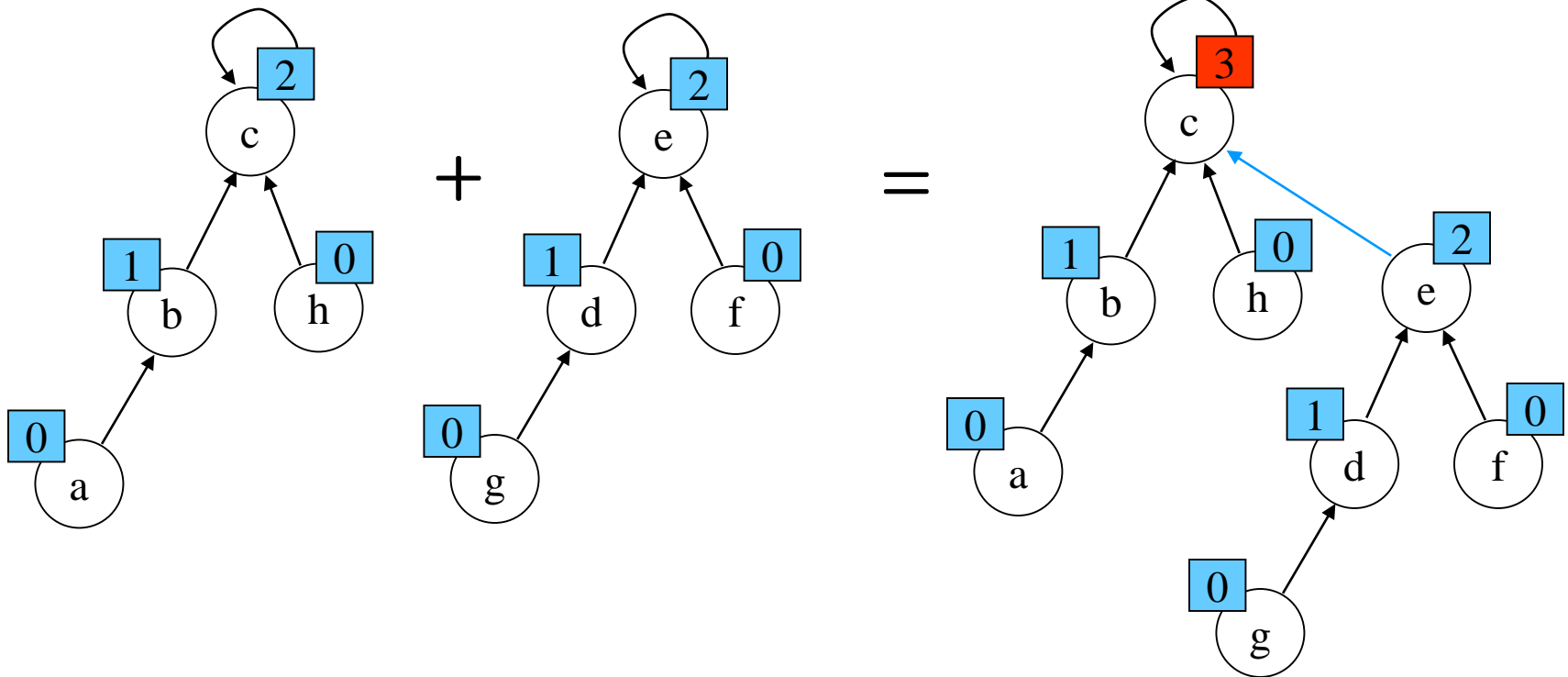
연산의 효율을 높이는 방법

- 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

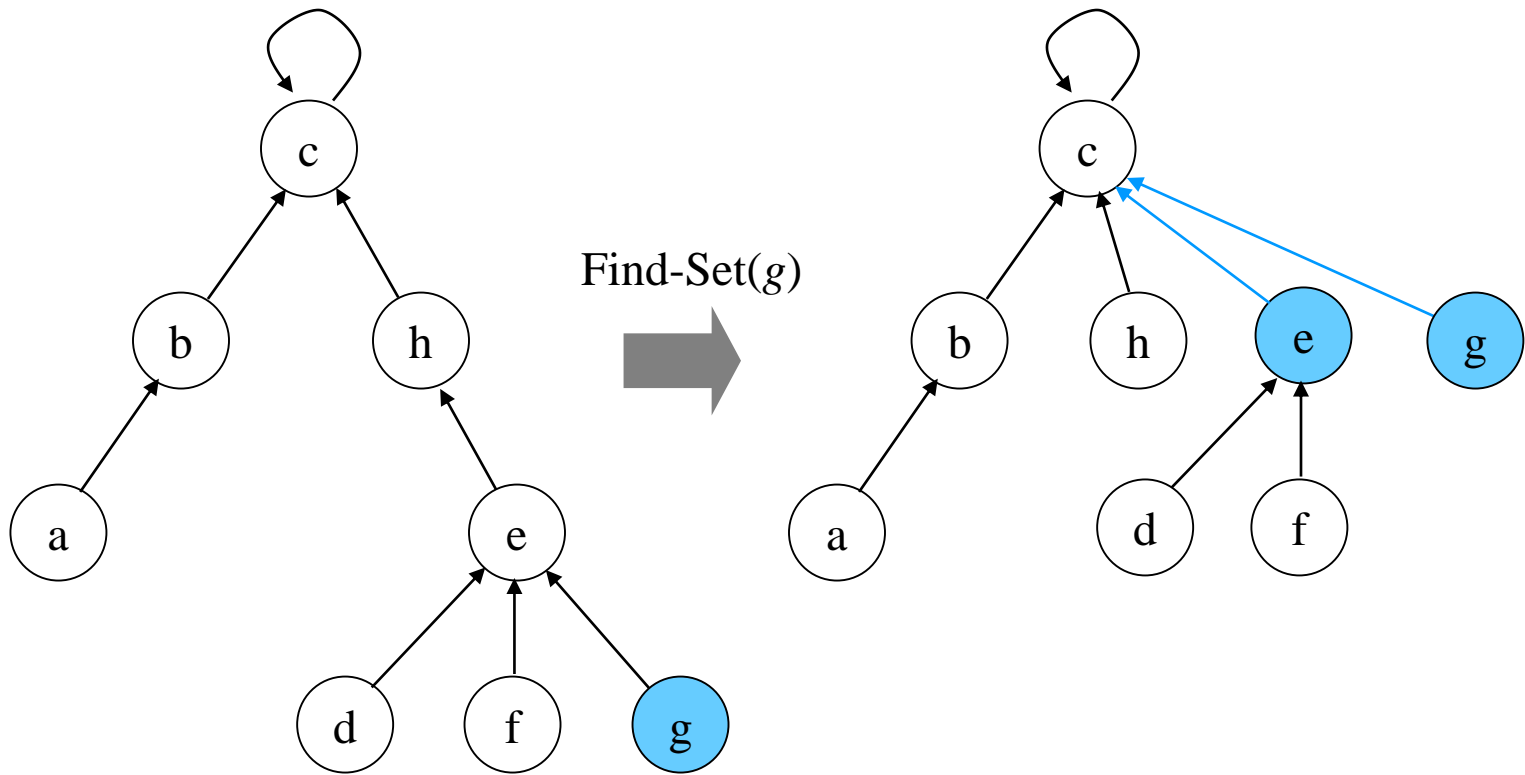
랭크를 이용한 Union의 예



랭크를 이용한 Union에서 랭크가 증가하는 예



Path Compression의 예



Rank를 이용한 Union과 Make-Set

Make-Set(x)

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

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  $\text{rank}[y'] \leftarrow \text{rank}[y'] + 1$ ;  
    }  
}
```

Path Compression을 이용한 Find-Set

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

[정리 5]

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

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

사실상 linear time임



Thank you
