8. Disjoint Sets

Goals

- Learn linked-list implementation for disjoint sets
- Learn tree implementation for disjoint sets
- Understand time complexities of two implementations

Disjoint Sets

- Consider data structures for disjoint sets.
- Intersection is not needed.
- Operations
  - Make-Set(x): create a set that contains only x
  - Find-Set(x): return the representative of the set containing x
  - Union(x, y): unite the set containing x and the set containing y
- Linked-list implementation and tree implementation

Finding Connected Components

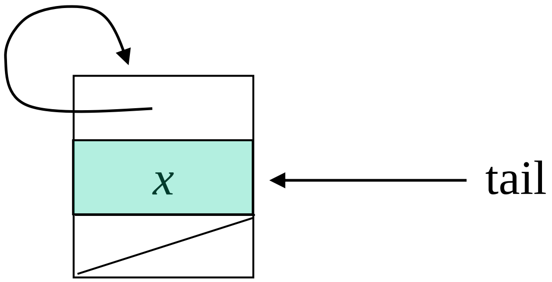
Connected-Components(G)
    for each vertex *v* in V(G)
        Make-Set(*v*)
    for each edge (*u*,*v*) in E(G)
        if Find-Set(*u*) $\neq$ Find-Set(*v*) **then** Union(*u*,*v*)

Same-Components(*u*,*v*)
    if Find-Set(*u*) = Find-Set(*v*) **then return** True
    else return False

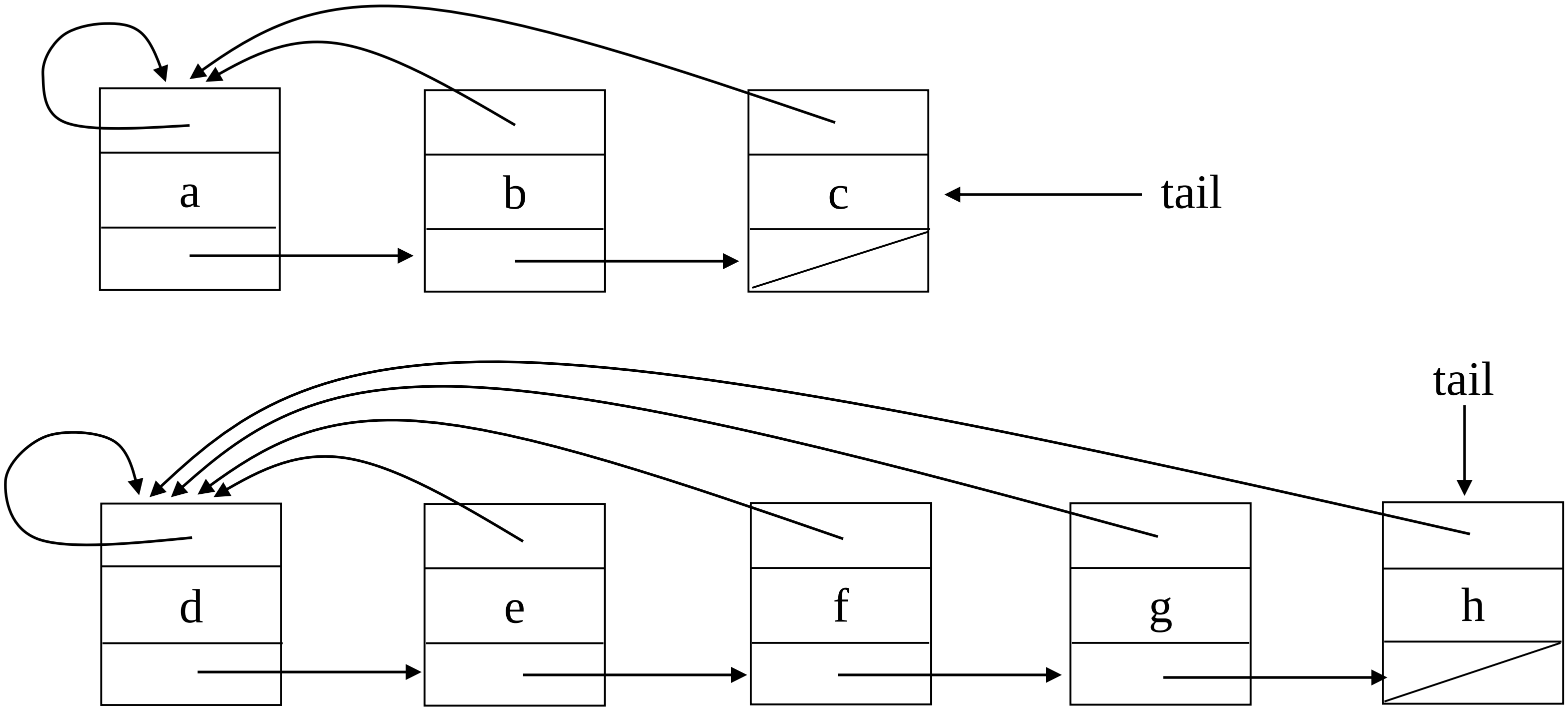
Linked-List Implementation

- Use a linked list for one set of elements
- The first element in a linked list is the representative of the set

Set containing one element

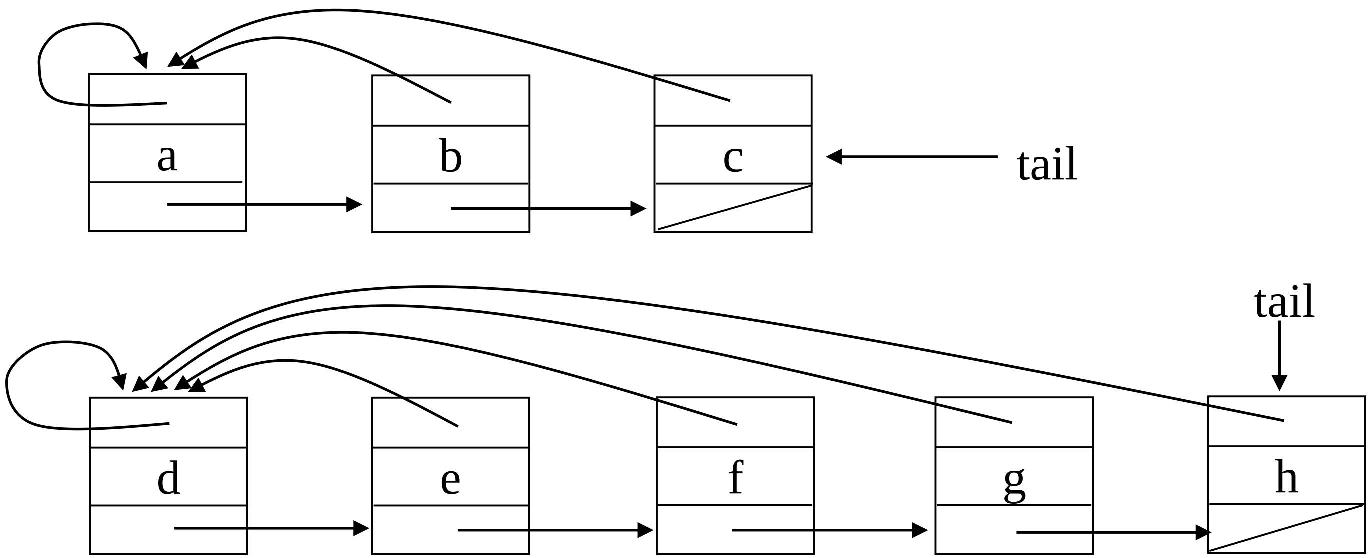


Two sets in linked-list implementation

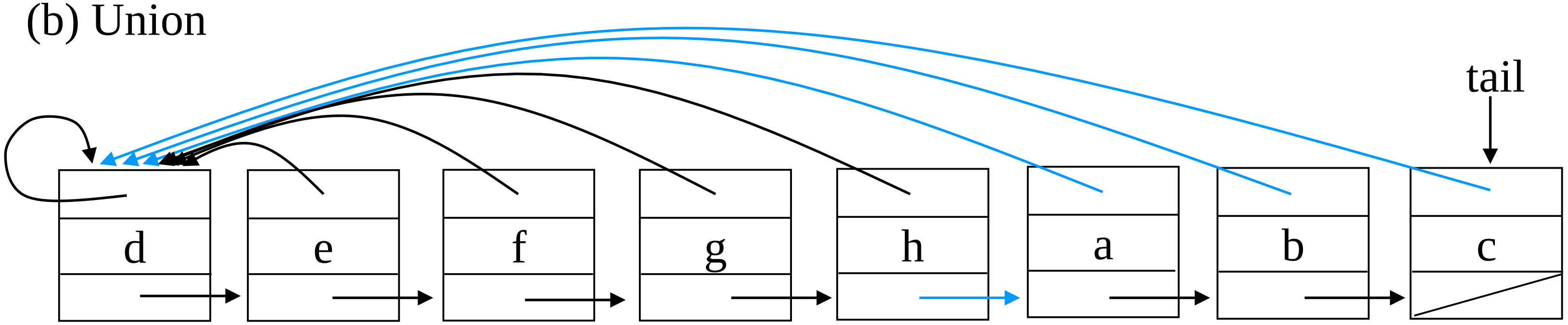


Union

(a) Two sets



(b) Union



Weighted Union

- When two linked lists are united, append the shorter list to the longer list
  - Minimize updates of pointers to the representative
- The representative of a set should have the weight of the set.

Time Complexity

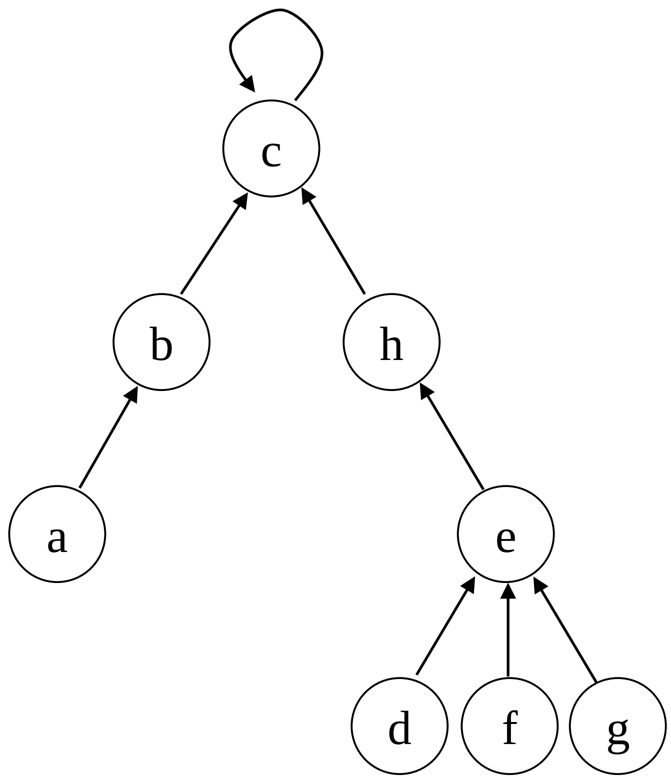
[Theorem 1]
When weighted union is used in linked-list implementation, a sequence of m Make-Set, Union, Find-Set operations, n of which are Make-Set operations, takes $O(m + n \log n)$ time.

[Proof] Make-Set, Find-Set: O(1) time
There are at most n-1 Union operations.
Time for Union: number of times pointers to representative are updated
Perspective of Union: $O(n^2)$
Perspective of elements: Whenever the pointer of x is updated, the size of the resulting set containing x doubles (at least).
Hence there are $\lceil \log n \rceil$ updates for x.
For all elements, $O(n \log n)$

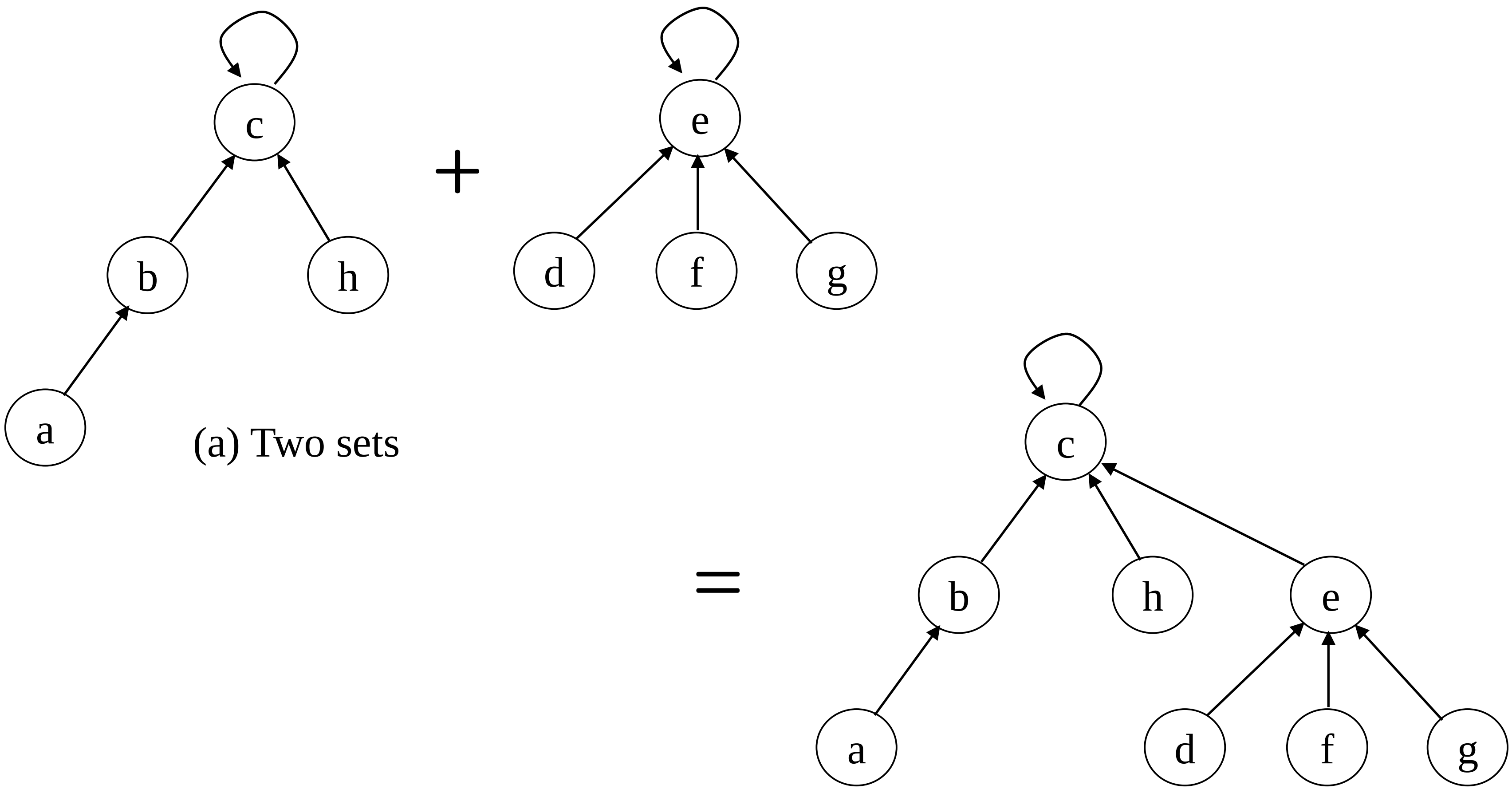
Tree Implementation

- Use a tree for one set of elements
  - A node has a pointer to its parent
- The root in a tree is the representative of the set

Tree Implementation



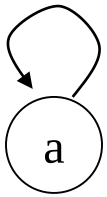
Union of two sets



(a) Two sets

(b) Union

Set containing one element



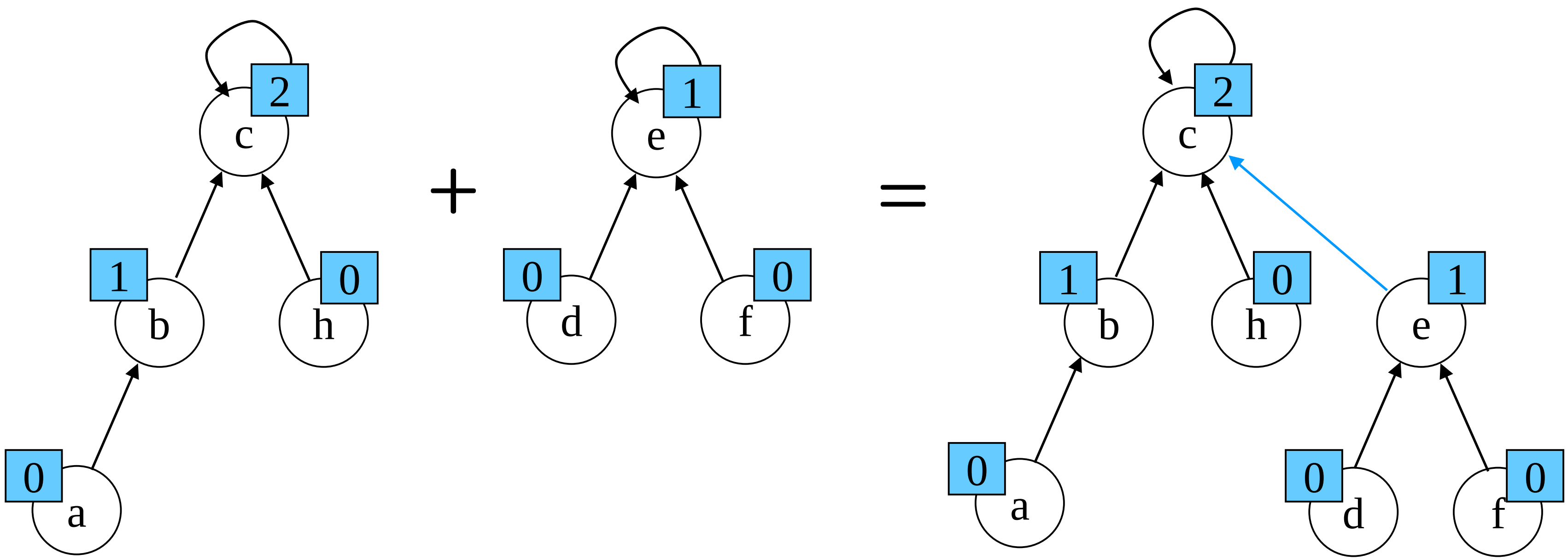
Operations in Tree Implementation

```
Make-Set(x)            ▷ Make a set containing only x
{
    p[x] ← x ;
}

Union(x, y)            ▷ Unite set containing x and set containing y
{
    p[Find-Set(y)] ← Find-Set(x) ;
}

Find-Set(x)            ▷ return representative of set containing x
{
    if (x  = p[x])
        then return x ;
        else return Find-Set(p[x]) ;
}
```

Two Heuristics to Improve Running Time

- Union by rank
  - Each node has a rank which is an upper bound on the height of the node
  - Make the root with smaller rank point to the root with larger rank
- Path compression
  - During Find-Set operation, make each node on the find path point directly to the root

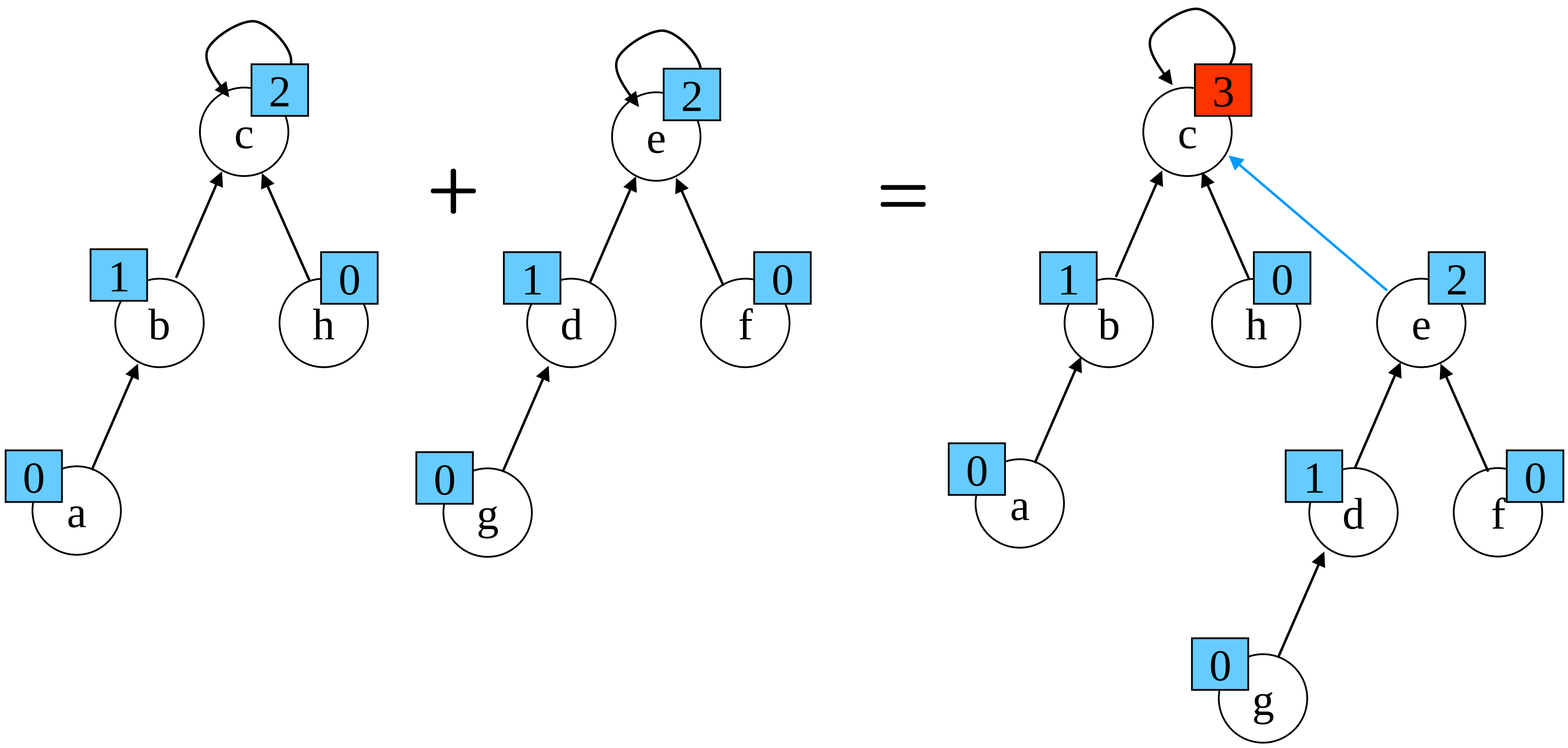
Union by Rank



(a) Two sets

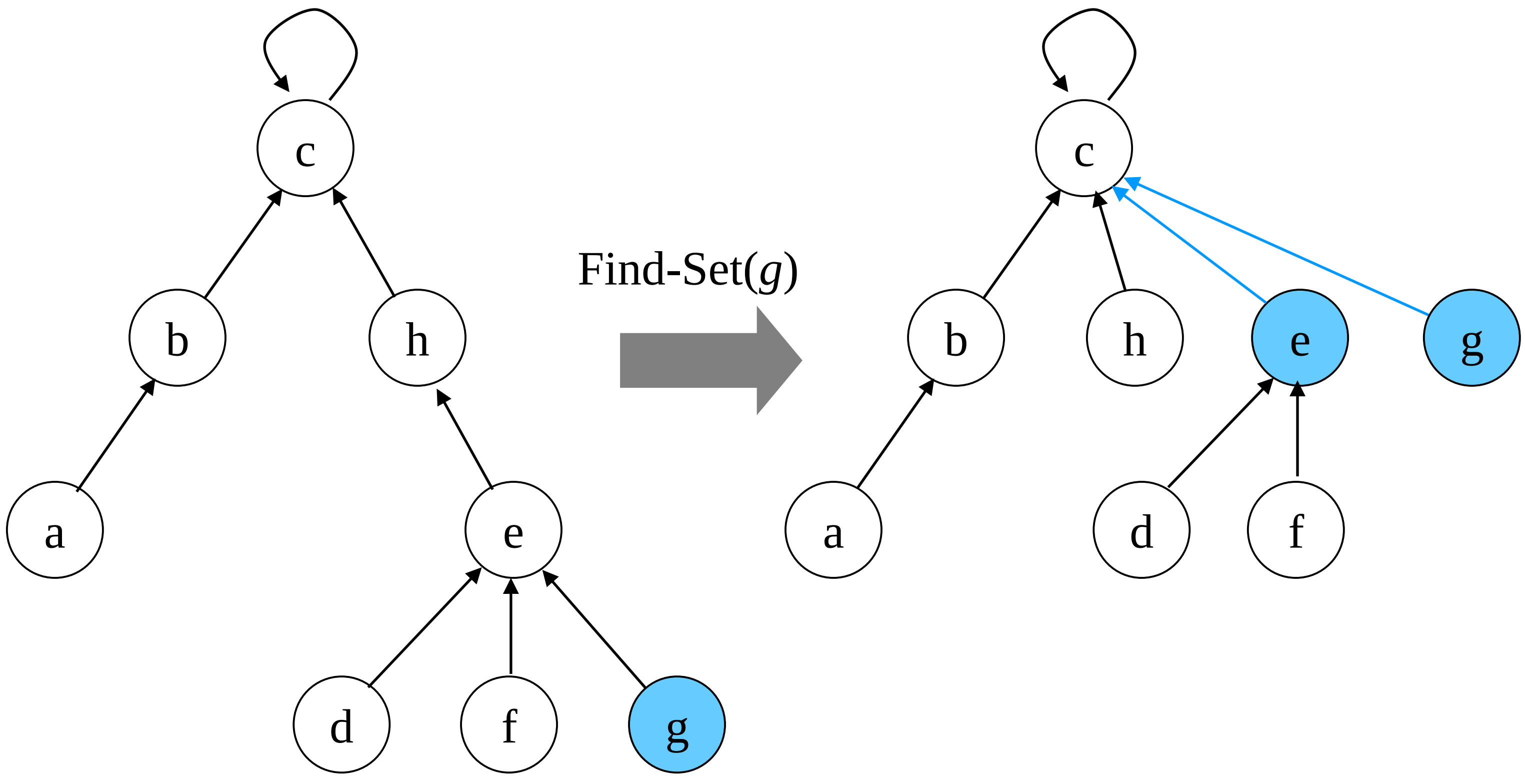
(b) Union

Union by Rank



(a) Two sets (b) Union

Path Compression



Make-Set and Union by Rank

```
Make-Set(x)            ▷ make a set containing only x
{
    p[x] ← x;
    rank[x] ← 0;
}

Union(x, y)            ▷ Unite set containing x and set containing y
{
    x' ← Find-Set(x);
    y' ← Find-Set(y);
    if (rank[x'] > rank[y'])
        then p[y'] ← x';
        else {
            p[x'] ← y';
            if (rank[x'] = rank[y']) then rank[y'] ← rank[y'] + 1;
        }
}
```

Find-Set with Path Compression

Find-Set(x)

$\triangleright$ return representative of set containing x

```
{
    if (p[x] ≠ x) then p[x] ← Find-Set(p[x]);
    return p[x];
}
```

Time Complexity

[Theorem 2]
When union by rank and path compression are used in tree implementation, a sequence of m Make-Set, Union, Find-Set operations, n of which are Make-Set operations, takes $O(m \log^* n)$ time.

$$\log^* n = \min\{k : \underbrace{\log \log \ldots \log}_{k \text{ times}} n \le 1\}$$

Almost linear time

Fischer $O(m \log \log n)$
Hopcroft and Ullman $O(m \log^* n)$
Tarjan $\Theta(m\alpha(n))$

Thank you