

Data Structures for Disjoint Sets

Heejin Park

Division of Computer Science and Engineering

Hanyang University

Contents

- **Disjoint-sets**
- **Disjoint-set operations**
- **An application of disjoint-set data structures**
- **Disjoint-set data structures**

Disjoint sets

• *Disjoint sets*

- Two sets A and B are disjoint if $A \cap B = \{\}$.

Ex> $A = \{1, 2\}, B = \{3, 4\}$

- Sets S_1, S_2, \dots, S_k are disjoint if every two distinct sets S_i and S_j are disjoint.

Ex> $S_1 = \{1, 2, 3\}, S_2 = \{4, 8\}, S_3 = \{5, 7\}$

Disjoint sets

- A *collection* of disjoint sets

- A set of disjoint sets is called a collection of disjoint sets.

Ex> $\{\{1, 2, 3\}, \{4, 8\}, \{5, 7\}\}$

- Each set in a collection has a *representative member* and the set is identified by the member.

Ex> $\{\{1, 2, 3\}, \{4, 8\}, \{5, 7\}\}$

Disjoint sets

- A collection of *dynamic disjoint sets*
 - **Dynamic:** Sets are changing.
 - New sets are created.
 - $\{\{1, 2, 3\}, \{4, 8\}, \{5, 7\}\} \rightarrow \{\{1, 2, 3\}, \{4, 8\}, \{5, 7\}, \{9\}\}$
 - Two sets are united.
 - $\{\{1, 2, 3\}, \{4, 8\}, \{5, 7\}\} \rightarrow \{\{1, 2, 3\}, \{4, 8, 5, 7\}\}$

Disjoint-set operations

• Disjoint-set operations

- **MAKE-SET(x)**
- **UNION(x, y)**
- **FIND-SET(x)**

Disjoint-set operations

• MAKE-SET(x)

- Given a member x , generate a set for x .
- MAKE-SET(9)

$\{\{1, 2, 3\}, \{4, 8\}, \{5, 7\}\} \rightarrow \{\{1, 2, 3\}, \{4, 8\}, \{5, 7\}, \{9\}\}$

Disjoint-set operations

• **UNION(x, y)**

- Given two members x and y , unite the set containing x and another set containing y .
- UNION(1,4)
- $\{\{1, 2, 3\}, \{4, 8\}, \{5, 7\}\} \rightarrow \{\{1, 2, 3, 4, 8\}, \{5, 7\}\}$

• **FIND-SET(x)**

- Find the representative of the set containing x .
- FIND-SET(5): 7

Disjoint-set data structures

● Problem

- *Developing data structures* to maintain a collection of dynamic disjoint sets supporting disjoint-set operations, which are MAKE-SET(x), UNION(x,y), FIND-SET(x).

Disjoint-set data structures

Parameters for running time analysis

- #Total operations: m
- #MAKE-SET ops: n
- #UNION ops: u
- #FIND-SET ops: f
- $m = n + u + f$

Disjoint-set data structures

• $u \leq n - 1$

- n is the number of sets are generated by MAKE-SET ops.
- Each UNION op reduces the number of sets by 1.
- So, after $n-1$ UNION ops, we have only 1 set and then we cannot do UNION op more.

Assumption

- The first n operations are MAKE-SET operations.

Contents

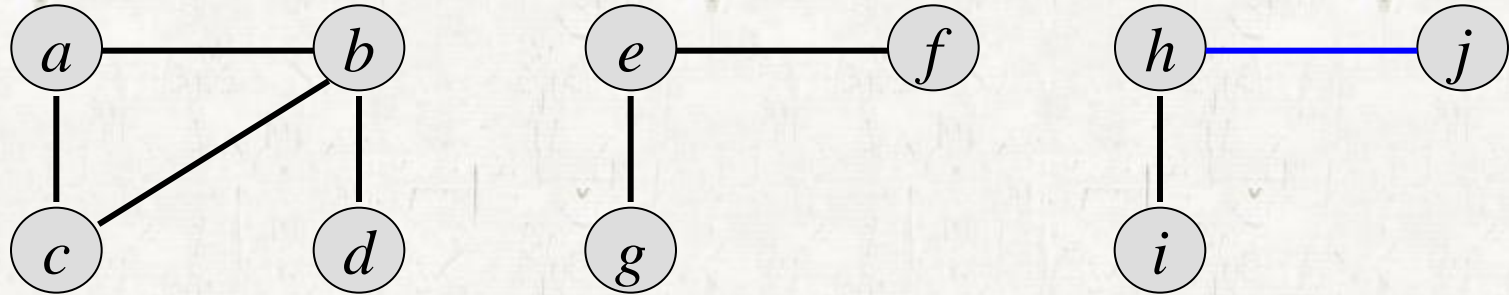
- Disjoint-sets
- Disjoint-set operations
- **An application of disjoint-set data structures**
- **Disjoint-set data structures**

Application

• Computing connected components (CC)

- Static graph
 - Depth-first search: $\Theta(V+E)$
- Dynamic graph
 - Depth-first search is inefficient.
 - Maintaining a disjoint-set data structure is more efficient.

Connected component computation



$\{\{a,b,c,d\}, \{e,f,g\}, \{h,i\}, \{j\}\}$

$\rightarrow \{\{a,b,c,d\}, \{e,f,g\}, \{h,i,j\}\}$

Depth first search: $\Theta(V + E)$

Disjoint-set data structures: $\text{UNION}(h, j)$

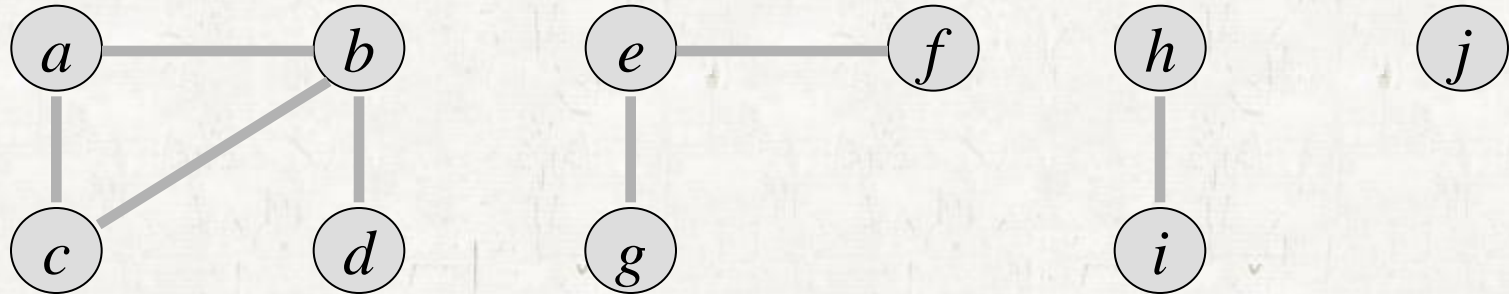
Connected component computation

• Computing CC using disjoint set operations

CONNECTED-COMPONENTS(G)

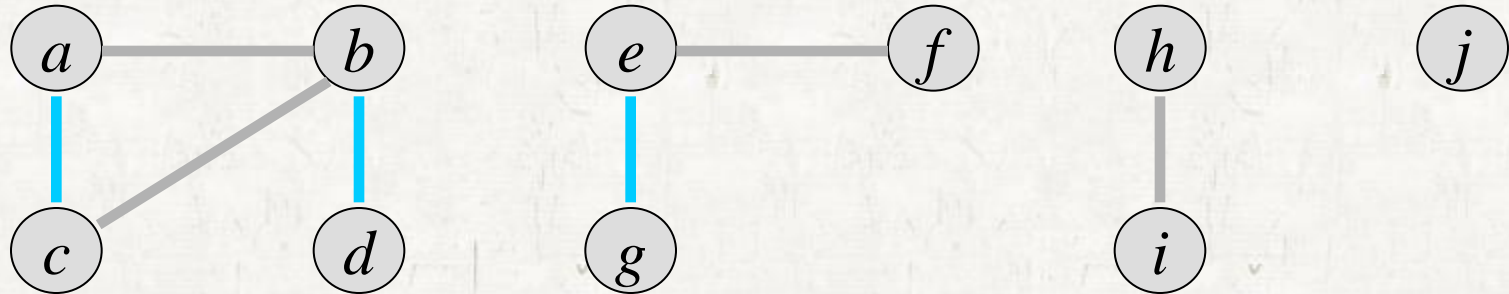
```
1  for each vertex  $v \in G.V$ 
2      MAKE-SET( $v$ )
3  for each edge  $(u, v) \in G.E$ 
4      if FIND-SET( $u$ )  $\neq$  FIND-SET( $v$ )
5          UNION( $u, v$ )
```

Connected component computation



Initial sets	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(b,d)	$\{a\}$	$\{b,d\}$	$\{c\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$	
(e,g)	$\{a\}$	$\{b,d\}$	$\{c\}$	$\{e,g\}$	$\{f\}$	$\{h\}$	$\{i\}$	$\{j\}$		
(a,c)	$\{a,c\}$	$\{b,d\}$		$\{e,g\}$	$\{f\}$	$\{h\}$	$\{i\}$	$\{j\}$		

Connected component computation



(a,c)	$\{a,c\}$	$\{b,d\}$	$\{e,g\}$	$\{f\}$	$\{h\}$	$\{i\}$	$\{j\}$
(h,i)	$\{a,c\}$	$\{b,d\}$	$\{e,g\}$	$\{f\}$	$\{h,i\}$		$\{j\}$
(a,b)	$\{a,b,c,d\}$		$\{e,g\}$	$\{f\}$	$\{h,i\}$		$\{j\}$
(e,f)	$\{a,b,c,d\}$		$\{e,f,g\}$		$\{h,i\}$		$\{j\}$
(b,c)	$\{a,b,c,d\}$		$\{e,f,g\}$		$\{h,i\}$		$\{j\}$

Connected component computation

SAME-COMPONENT(u, v)

```
1  if FIND-SET( $u$ ) == FIND-SET( $v$ )  
2      return TRUE  
3  else return FALSE
```

Contents

- Disjoint-sets
- Disjoint-set operations
- An application of disjoint-set data structures
- **Disjoint-set data structures**

Disjoint-set data structures

• Disjoint-set data structures

- Linked-list representation
- Forest representation

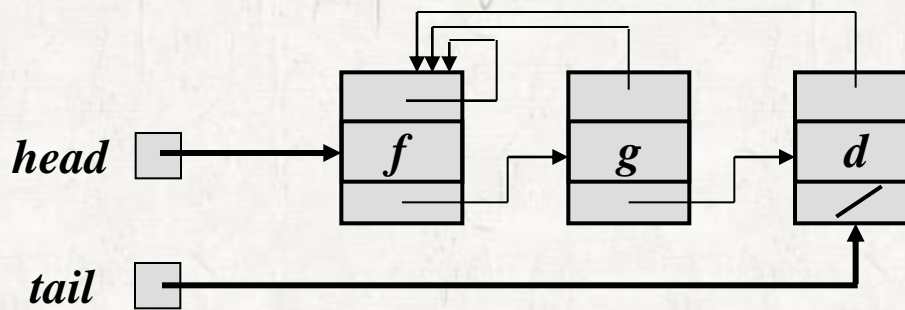
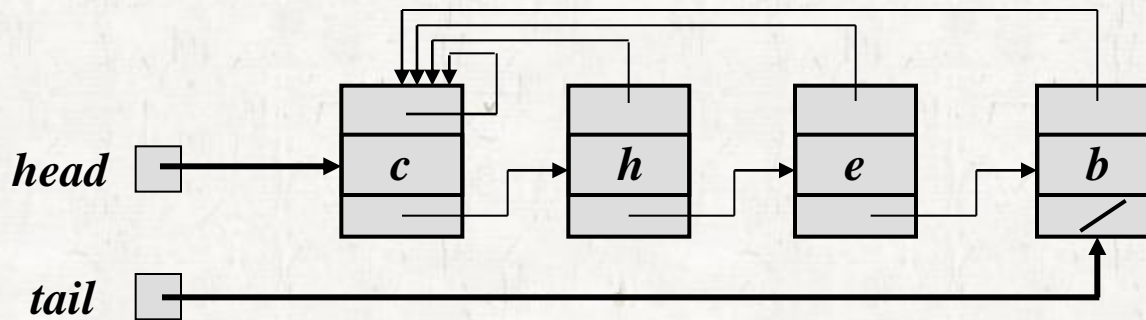
Linked-list representation

◉ Linked-list representation

- Each set is represented by a linked list.
- Members of a disjoint set are objects in a linked list.
- The first object in the linked list is the representative.
- All objects have pointers to the representative.

Linked-list representation

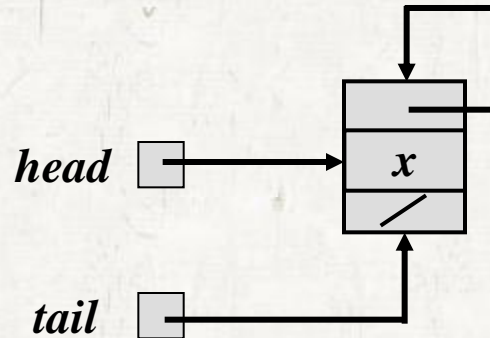
$\{\{b,c,e,h\}, \{d,f,g\}\}$: Two linked lists are needed.



Linked-list representation

- **MAKE-SET(x)**

- $\Theta(1)$

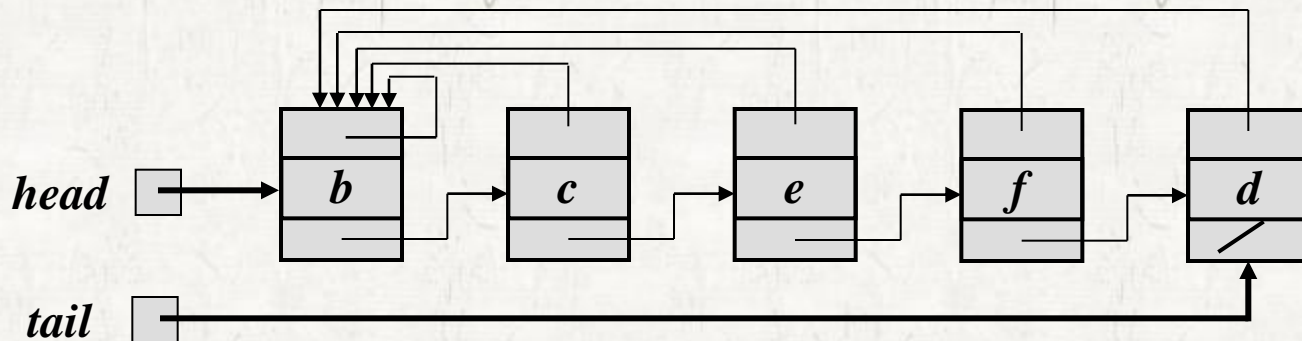
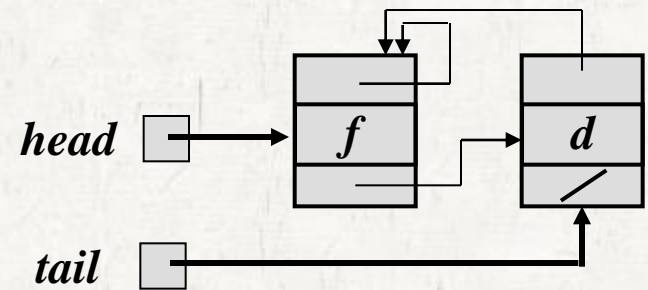
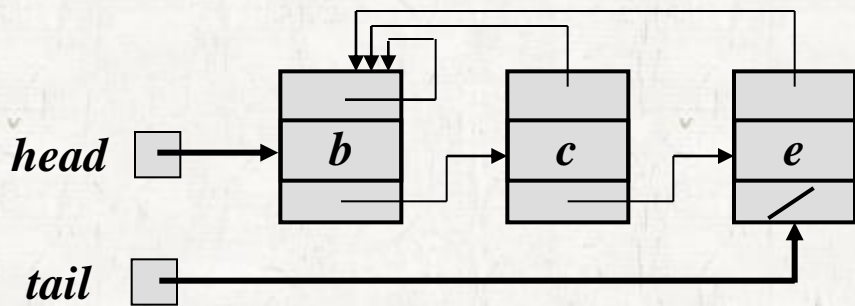


- **FIND-SET(x)**

- $\Theta(1)$

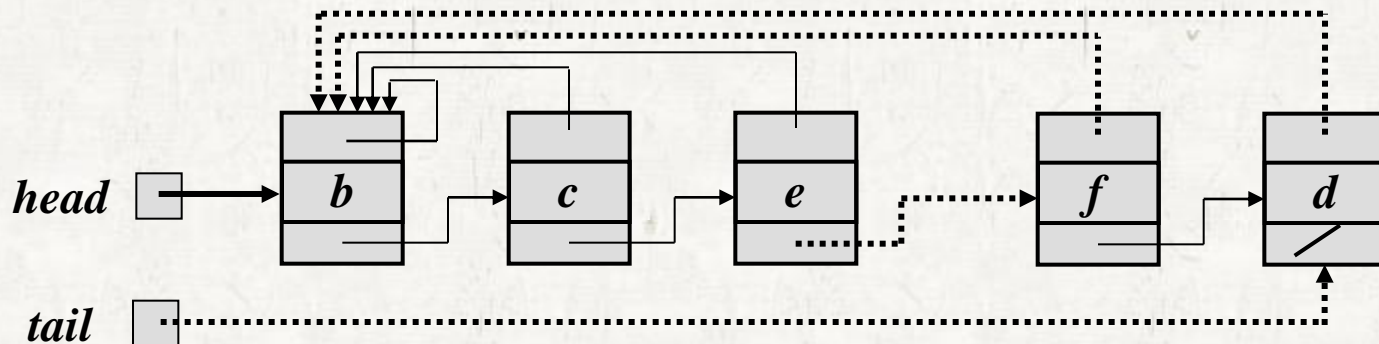
Linked-list representation

- **UNION(x,y):** Attaching a linked list to the other



Linked-list representation

- **UNION(x,y):** Attaching a linked list to the other



- $\Theta(m_2)$ time where m_2 is the number of objects in the linked list being attached.
- Changing tail pointer & linking two linked lists: $\Theta(1)$
- Changing pointers to the representative: $\Theta(m_2)$

Linked-list representation

• Running time for $m (= n + f + u)$ operations

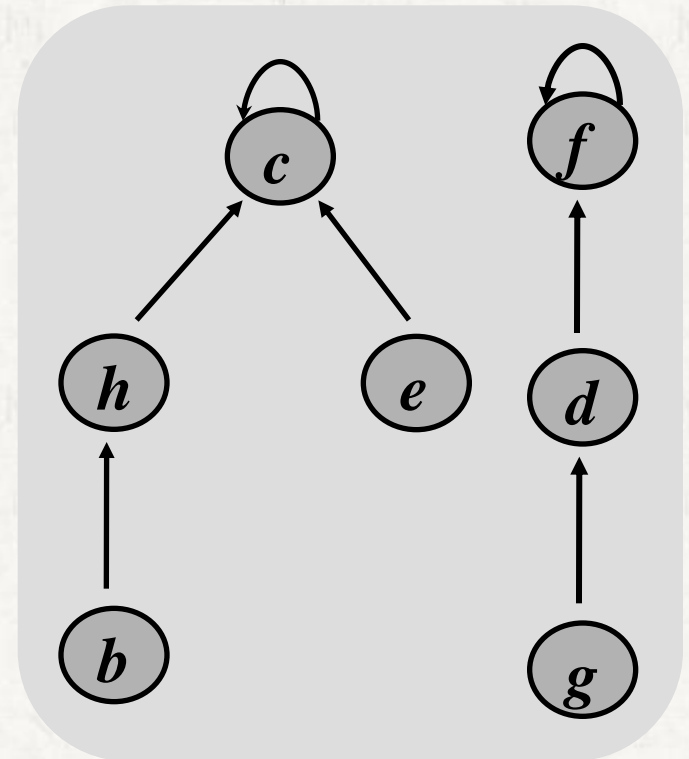
- Simple implementation of union
 - $O(n+f+u^2)$ time $\rightarrow O(m+n^2)$ time
 - Because $u < n$
- A weighted-union heuristic
 - $O(n+f+u \lg u)$ time $\rightarrow O(m+n \lg n)$ time

Forest representation

Forest representation

- Each set is represented by a tree.
- Each member points to its parent.
- The root of each tree is the rep.

$\{\{b,c,e,h\}, \{f,d,g\}\}$



Forest representation

MAKE-SET(x)

1 $x.p = x$

FIND-SET(x)

1 **if** $x == x.p$

2 **return** x

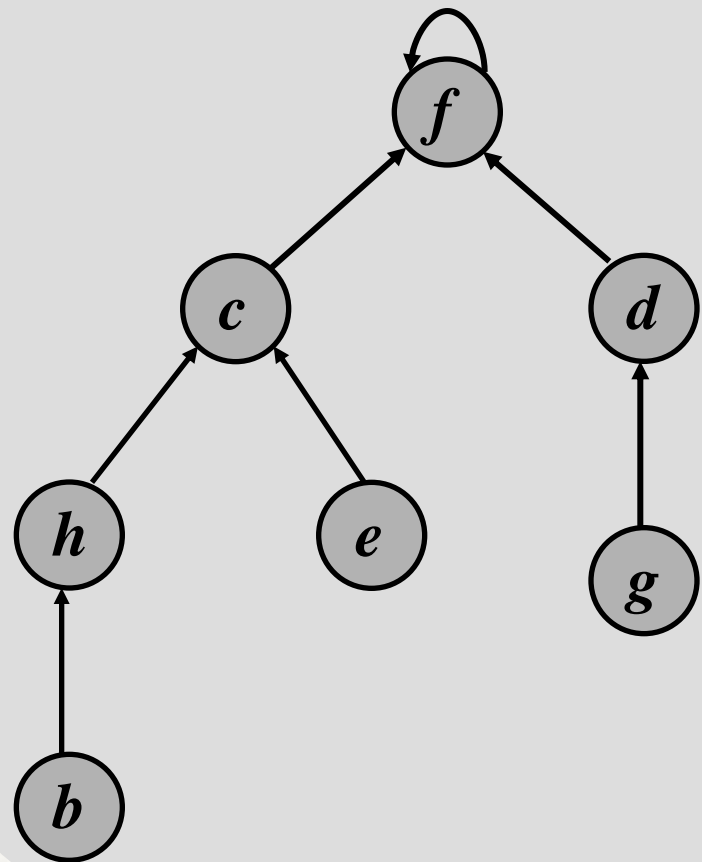
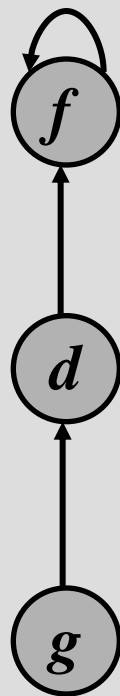
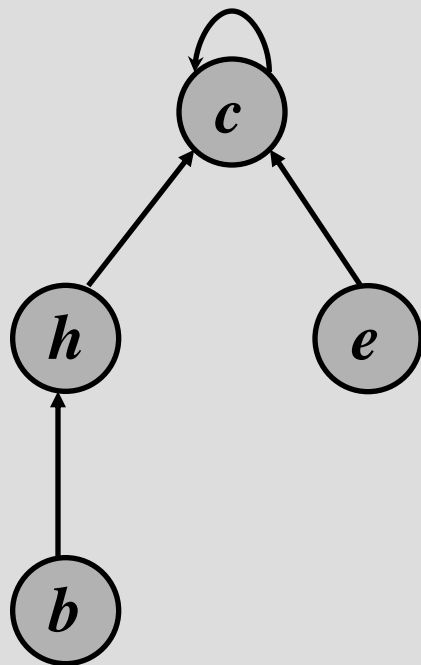
3 **else return** FIND-SET($x.p$)

Forest representation

• Union by rank

- *Idea:* Attach the shorter tree to the higher tree.
- Each node maintains a *rank*, which is an upper bound on the height of the node.
- Compare the ranks of the two roots and attach the tree whose root's rank is smaller to the other.

Forest representation



Forest representation

MAKE-SET(x)

```
1   $x.p = x$   
2   $x.rank = 0$ 
```

UNION(x, y)

```
1  LINK(FIND-SET( $x$ ), FIND-SET( $y$ ))
```

LINK(x, y)

```
1  if  $x.rank > y.rank$   
2       $y.p = x$   
3  else  $x.p = y$   
4      if  $x.rank == y.rank$   
5           $y.rank = y.rank + 1$ 
```

Forest representation

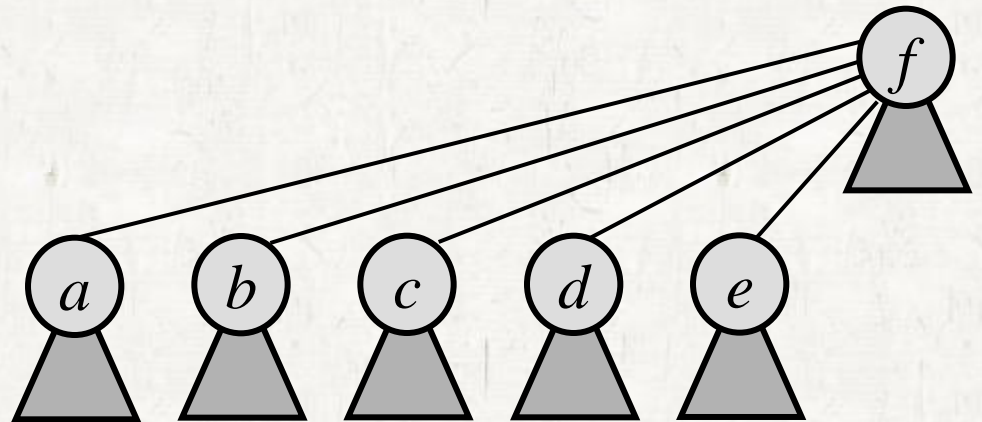
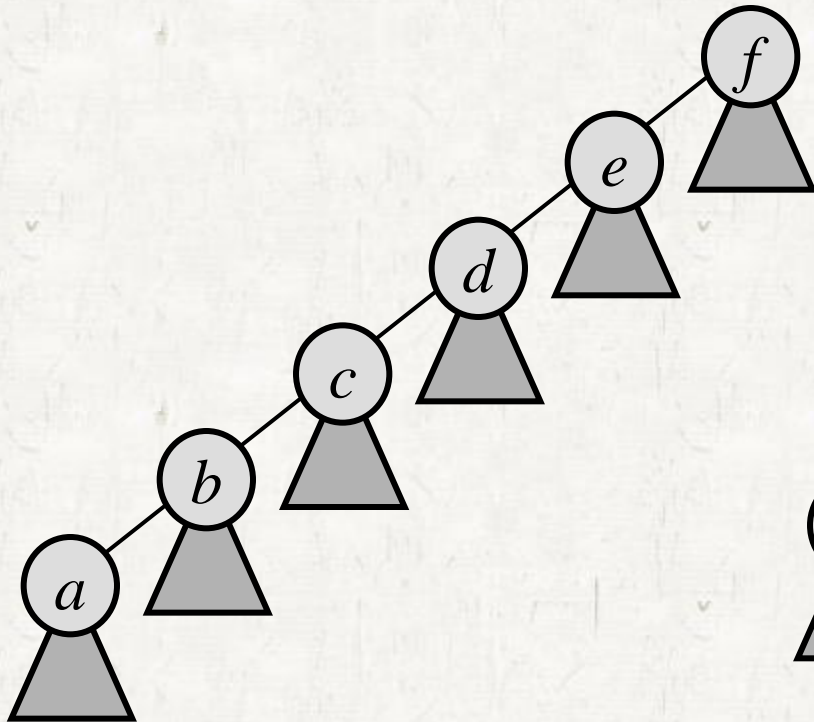
• Path compression

- Change the parent to the root during FIND-SET(x).

FIND-SET(x)

```
1  if  $x \neq x.p$   
2       $x.p = \text{FIND-SET}(x.p)$   
3  return  $x.p$ 
```


Forest representation



Forest representation

- Worst case running time : $O(m \alpha(n))$
- $\alpha(n) \leq 4$:for all practical situations.