CSE 2215: Data Structures and Algorithms-I

Data Structures for Disjoint Sets

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Vice Chancellor, UIU

Disjoint Sets

- Some applications require maintaining a collection of disjoint sets.

- A Disjoint Set ***S*** is a collection of sets $S_1, \ldots\ldots S_n$
  where $\forall_{i \neq j} S_i \cap S_j = \phi$

- Each set has a representative which is a member of the set (usually the minimum if the elements are comparable)

Disjoint Set Operations

- Make-Set(x) – Creates a new set S_x where x is it's only element (and therefore it is the representative of the set).
  $O(1)$ time.

- Union(x, y) – Replaces S_x, S_y by $S_x \cup S_y$.
  One of the elements of $S_x \cup S_y$ becomes the representative of the new set.
  $O(\log n)$ time.

- Find(x) – Returns the representative of the set containing x
  $O(\log n)$ time.

Analyzing Operations

- We usually analyze a sequence of m operations, of which n of them are Make_Set operations, and m is the total of Make_Set, Find, and Union operations.

- Each union operation decreases the number of sets in the data structure, so there can not be more than n-1 Union operations.

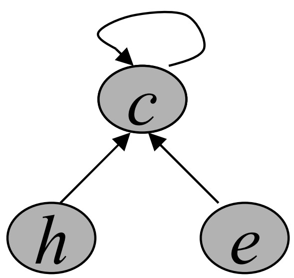
Applications

- Equivalence Relations (e.g Connected Components)

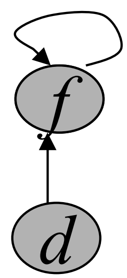
- Minimum Spanning Trees

Disjoint-Set Implementation: Forests

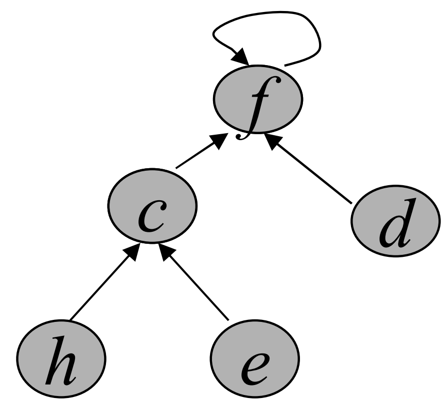
- Rooted trees, each tree is a set, root is the representative. Each node points to its parent. Root points to itself.



Set $\{c, h, e\}$



Set $\{f, d\}$



UNION

Straightforward Solution

- Three operations
  - MAKE-SET(x): create a tree containing x. Time: $O(1)$
  - FIND-SET(x): follow the chain of parent pointers until to the root. Time: $O(h)$, h is height of x's tree
  - UNION(x, y): let the root of one tree point to the root of the other. Time: $O(1)$

- It is possible that n-1 UNIONs result in a tree of height n-1. (just a linear chain of n nodes).

- So n FIND-SET operations will cost $O(n^2)$.

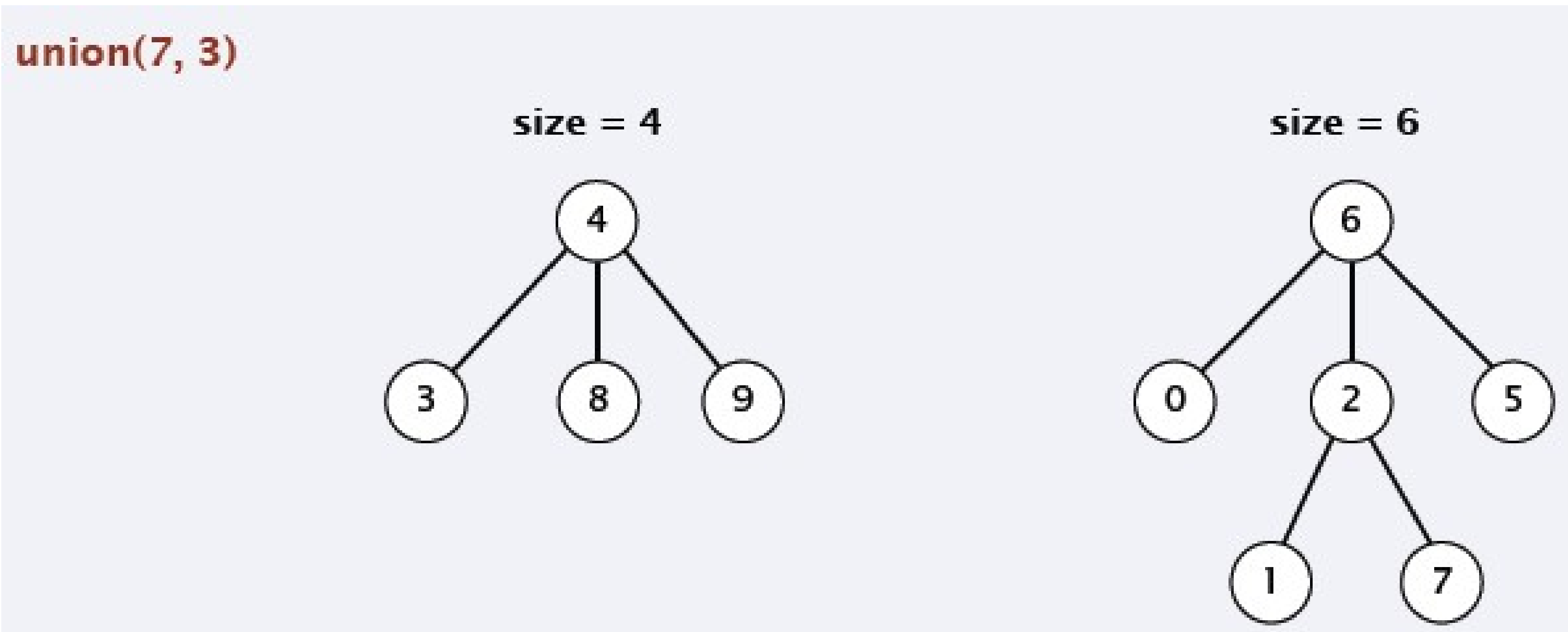
Union by Rank & Path Compression Heuristics

- Union by Rank: Each root is associated with a rank. Then when UNION, let the root with smaller rank point to the root with larger rank.
  - Link by Size, which is the number of nodes in the subtree rooted at the node
  - Link by Height, which is the height of the subtree rooted at the node

- Path Compression: used in FIND-SET(x) operation, make each node in the path from x to the root directly point to the root. Thus reduce the tree height.

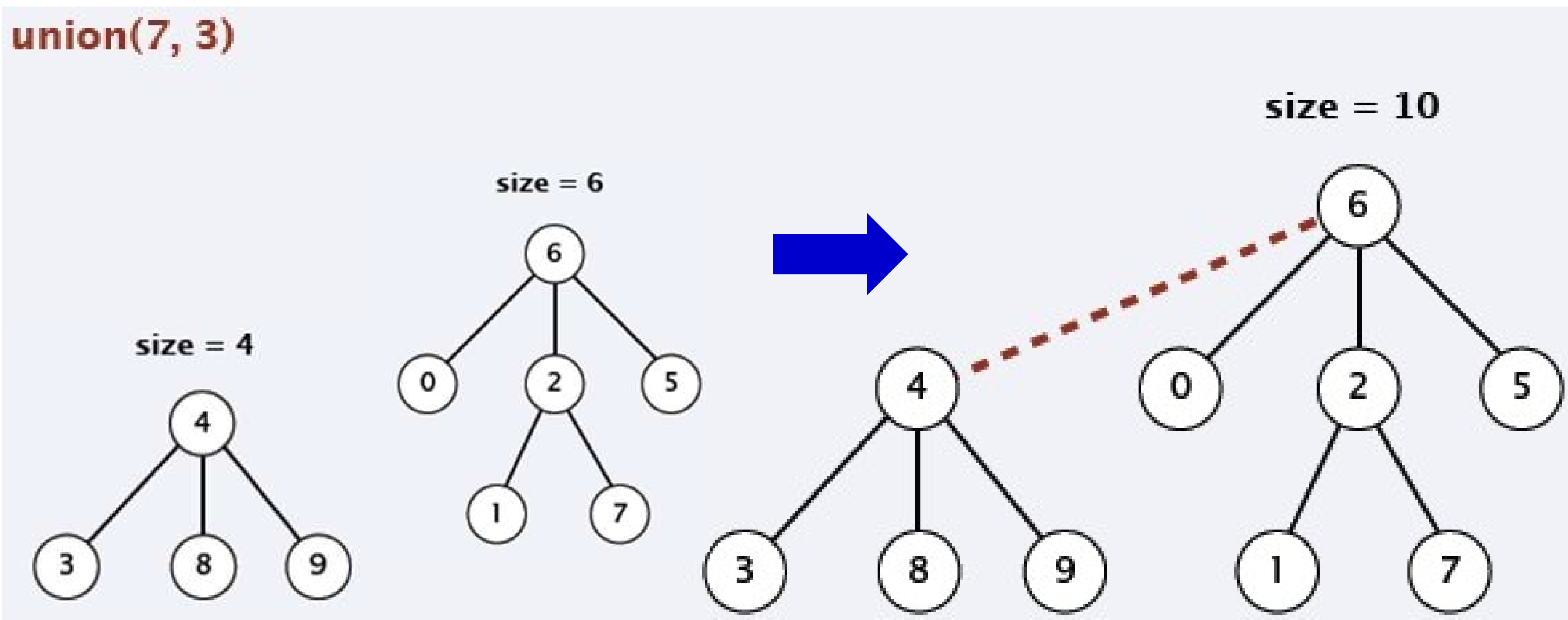
Link by Size

- Maintain a subtree count for each node, initially 1.
- Link root of smaller tree to root of larger tree (breaking ties arbitrarily).



Link by Size

- Maintain a subtree count for each node, initially 1.
- Link root of smaller tree to root of larger tree (breaking ties arbitrarily).



Link by Size

- Maintain a subtree count for each node, initially 1.
- Link root of smaller tree to root of larger tree (breaking ties arbitrarily).

MAKE-SET (x)

$parent(x) \leftarrow x.$
$size(x) \leftarrow 1.$

FIND (x)

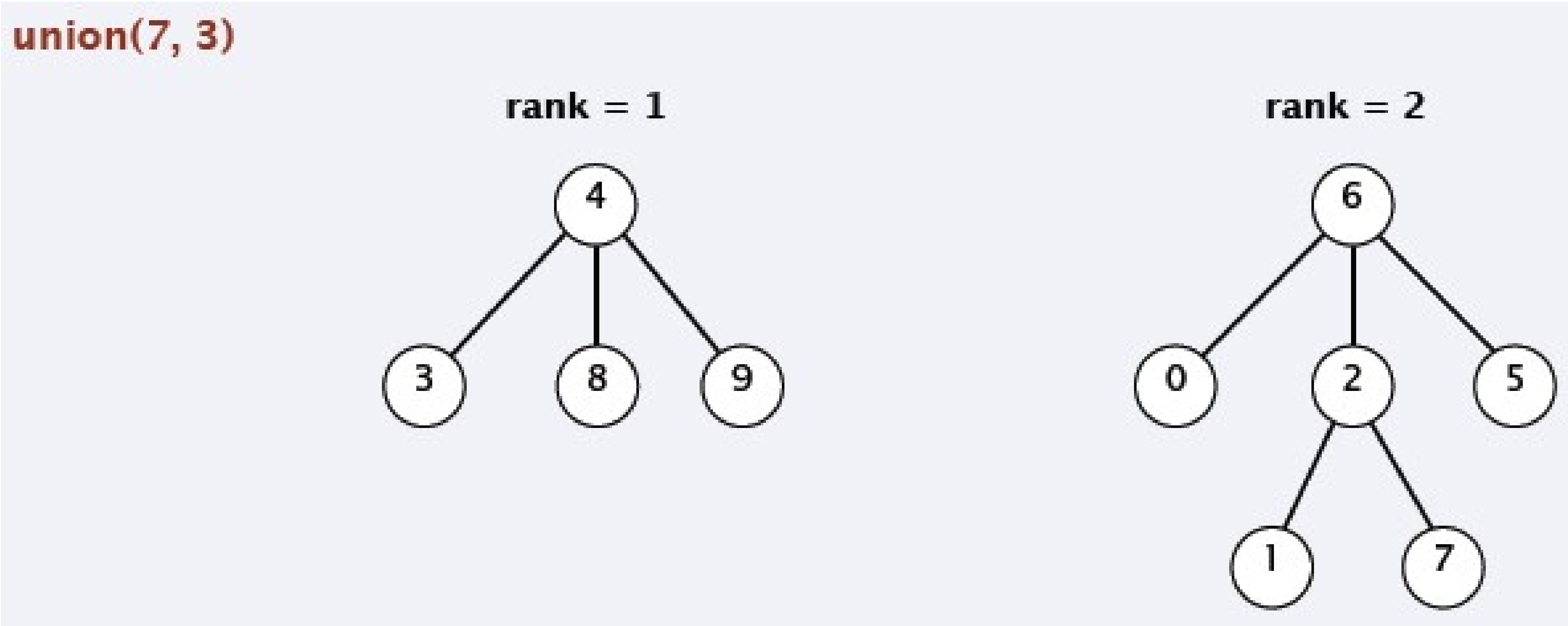
WHILE $(x \neq parent(x))$
    $x \leftarrow parent(x).$
RETURN $x.$

UNION-BY-SIZE (x, y)

$r \leftarrow$ **FIND** $(x).$
$s \leftarrow$ **FIND** $(y).$
IF $(r = s)$ **RETURN**.
ELSE IF $(size(r) > size(s))$
    $parent(s) \leftarrow r.$
    $size(r) \leftarrow size(r) + size(s).$
ELSE
    $parent(r) \leftarrow s.$
    $size(s) \leftarrow size(r) + size(s).$

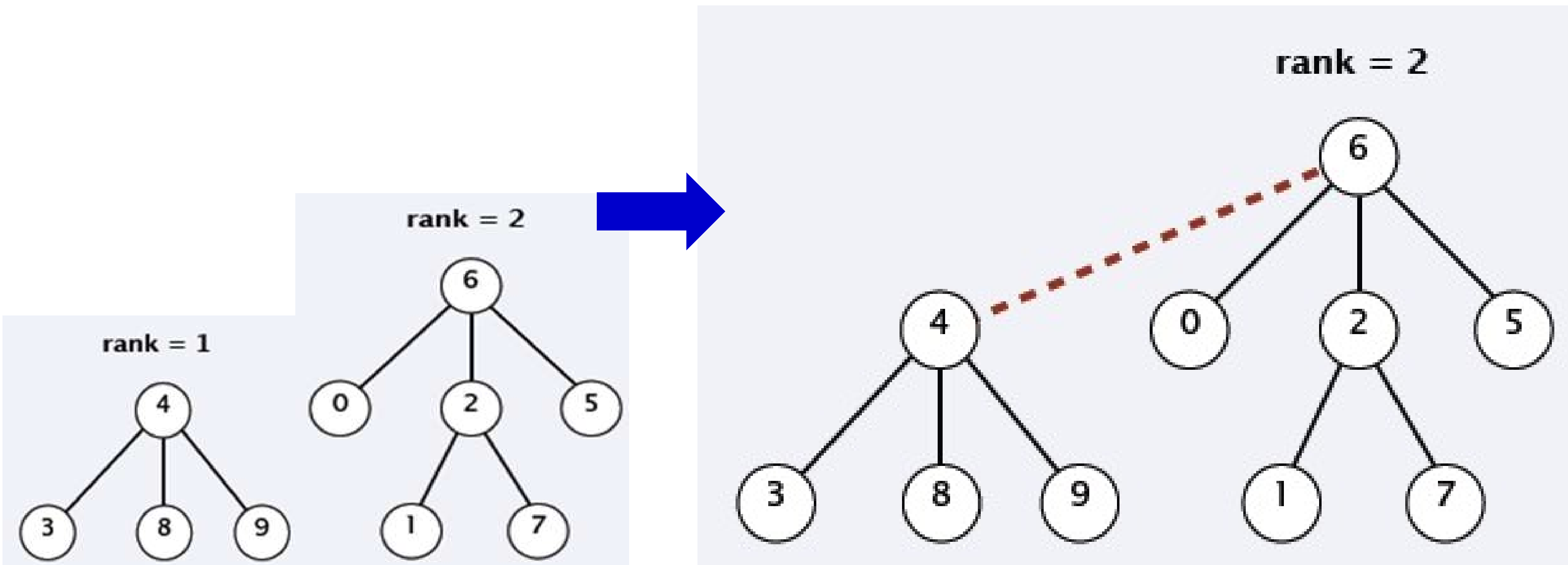
Link by Height

- Maintain an integer rank (height) for each node, initially 0.
- Link root of smaller rank (height) to root of larger rank (height); if tie, increase rank (height) of new root by 1.



Link by Height

- Maintain an integer rank (height) for each node, initially 0.
- Link root of smaller rank (height) to root of larger rank (height); if tie, increase rank (height) of new root by 1.



Link by Height

- Maintain an integer rank (height) for each node, initially 0.
- Link root of smaller rank (height) to root of larger rank (height); if tie, increase rank (height) of new root by 1.

MAKE-SET (x)

$parent(x) \leftarrow x.$

$rank(x) \leftarrow 0.$

FIND (x)

WHILE $x \neq parent(x)$

  $x \leftarrow parent(x).$

RETURN x.

UNION-BY-RANK (x, y)

$r \leftarrow$ FIND (x).

$s \leftarrow$ FIND (y).

IF $(r = s)$ RETURN.

ELSE IF $rank(r) > rank(s)$

  $parent(s) \leftarrow r.$

ELSE IF $rank(r) < rank(s)$

  $parent(r) \leftarrow s.$

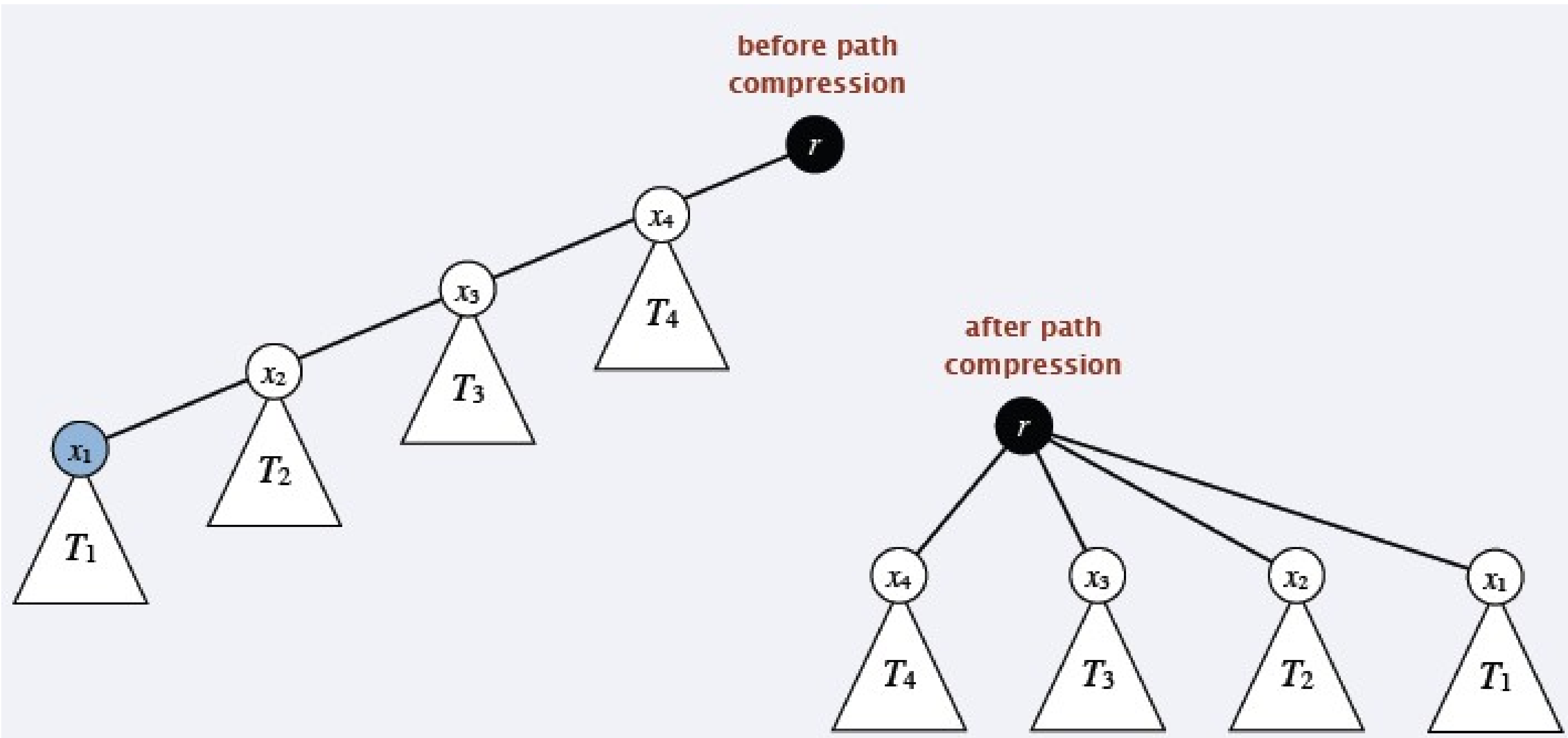
ELSE

  $parent(r) \leftarrow s.$

  $rank(s) \leftarrow rank(s) + 1.$

Path Compression

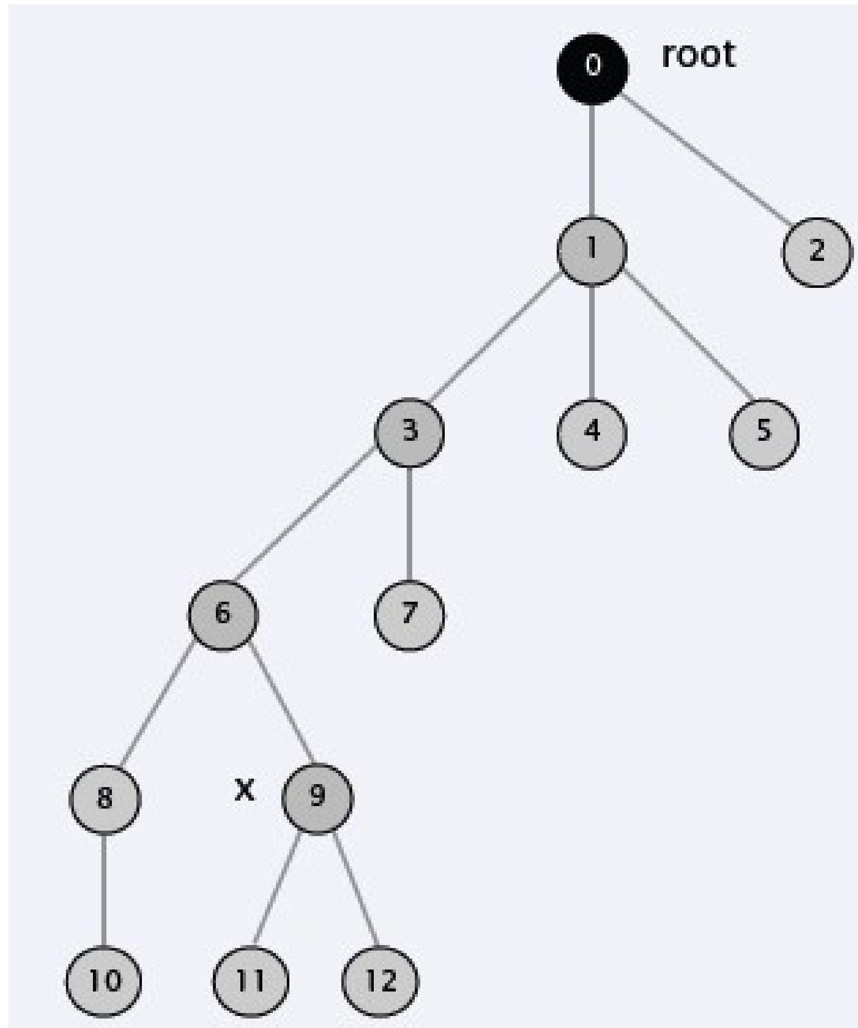
- After finding the root r of the tree containing x, change the parent pointer of all nodes along the path to point directly to r.



Path compression can cause a very deep tree to become very shallow

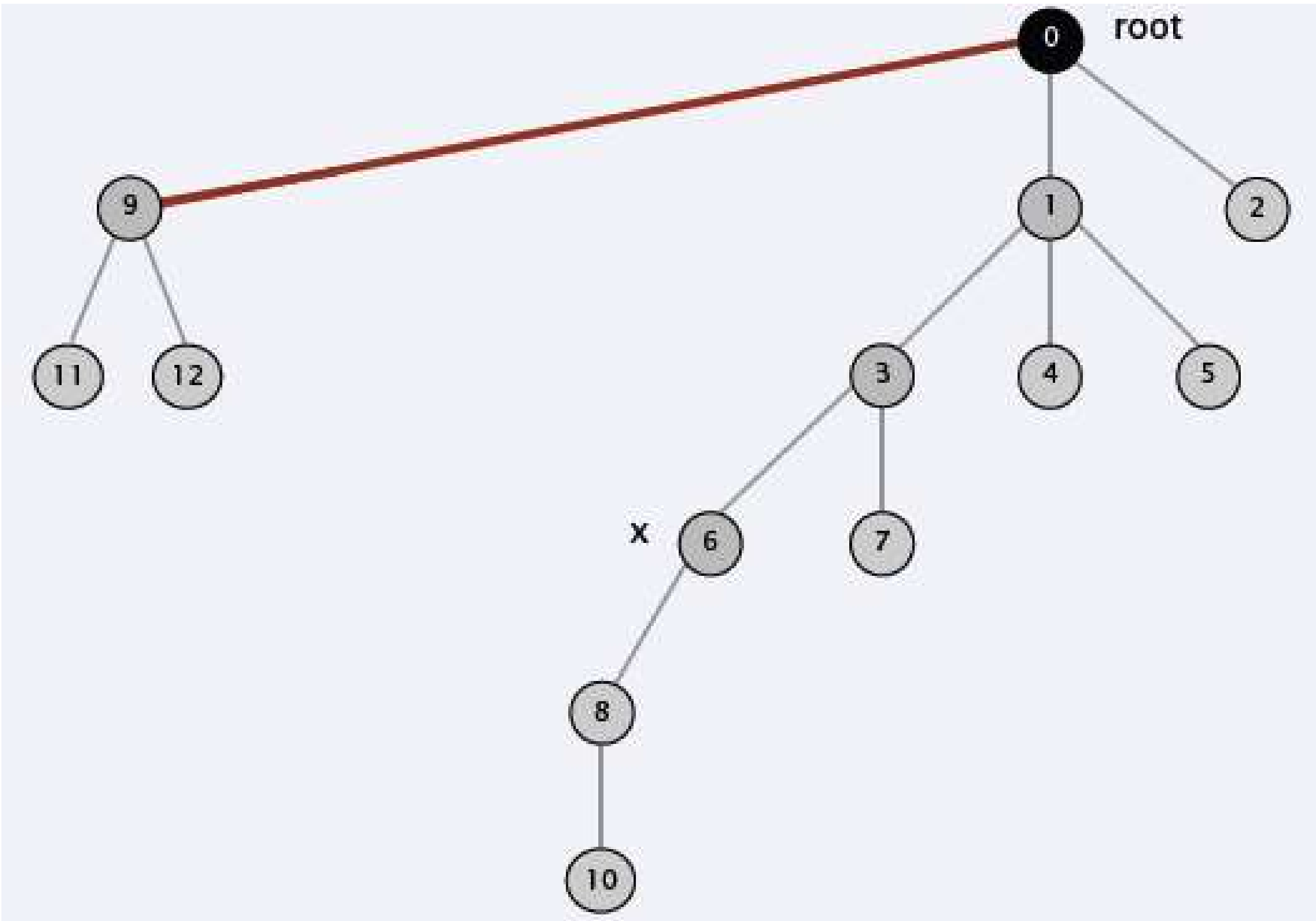
Path Compression

- After finding the root *r* of the tree containing *x*, change the parent pointer of all nodes along the path to point directly to *r*.



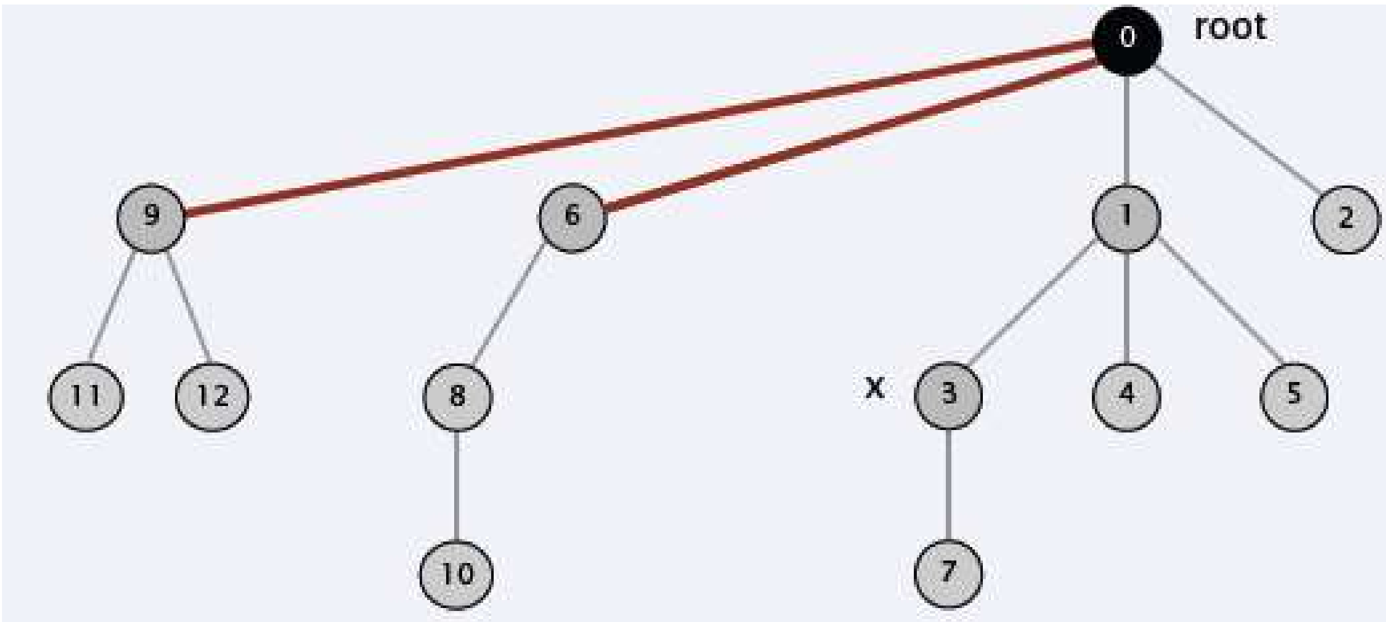
Path Compression

- After finding the root r of the tree containing x, change the parent pointer of all nodes along the path to point directly to r.



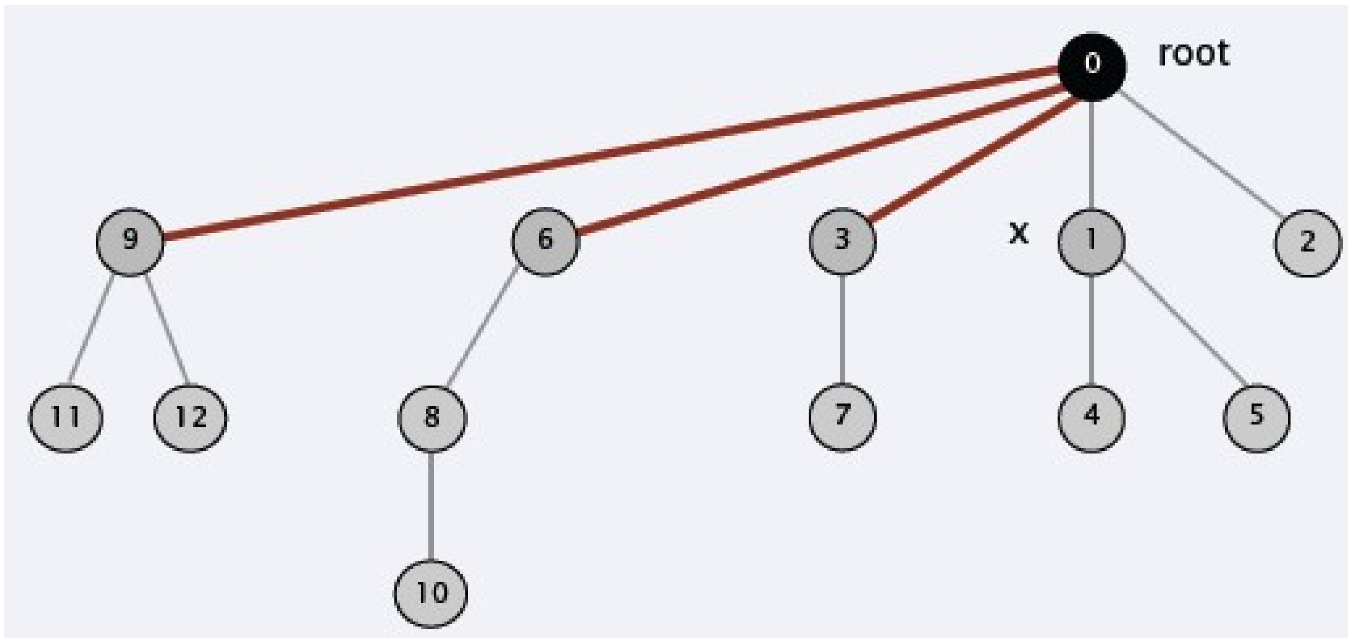
Path Compression

- After finding the root r of the tree containing x, change the parent pointer of all nodes along the path to point directly to r.



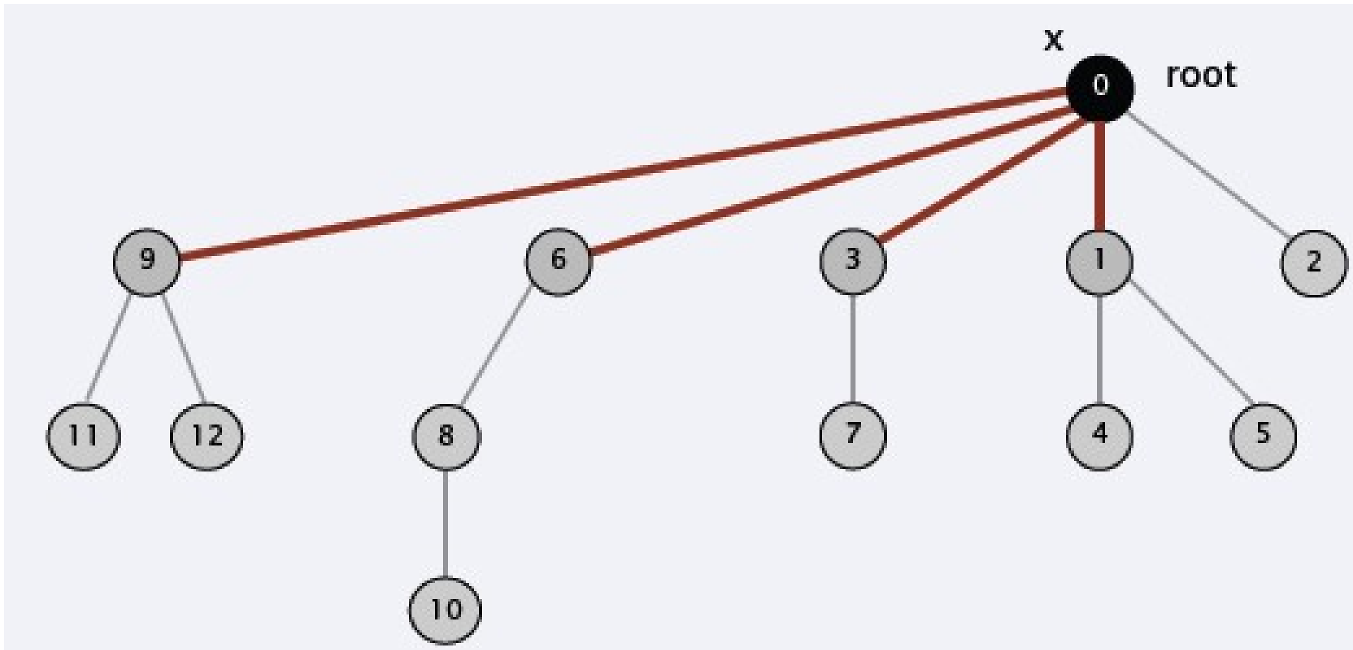
Path Compression

- After finding the root r of the tree containing x, change the parent pointer of all nodes along the path to point directly to r.



Path Compression

- After finding the root r of the tree containing x, change the parent pointer of all nodes along the path to point directly to r.



Path compression can cause a very deep tree to become very shallow

Path Compression

- After finding the root r of the tree containing x, change the parent pointer of all nodes along the path to point directly to r.

FIND (x)

IF $x \neq parent(x)$

  $parent(x) \leftarrow$ FIND $(parent(x))$.

RETURN $parent(x)$.

Note: Path compression does not change the rank of a node;
So $height(x) \leq rank(x)$ but they are not necessarily equal.

Algorithm for Disjoint-Set Forest

MAKE-SET(x)
1. $p[x] \leftarrow x$
2. $rank[x] \leftarrow 0$

UNION(x, y)
1. LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
1. **if** $rank[x] > rank[y]$
2. **then** $p[y] \leftarrow x$
3. **else** $p[x] \leftarrow y$
4. **if** $rank[x] = rank[y]$
5. **then** $rank[y]$++

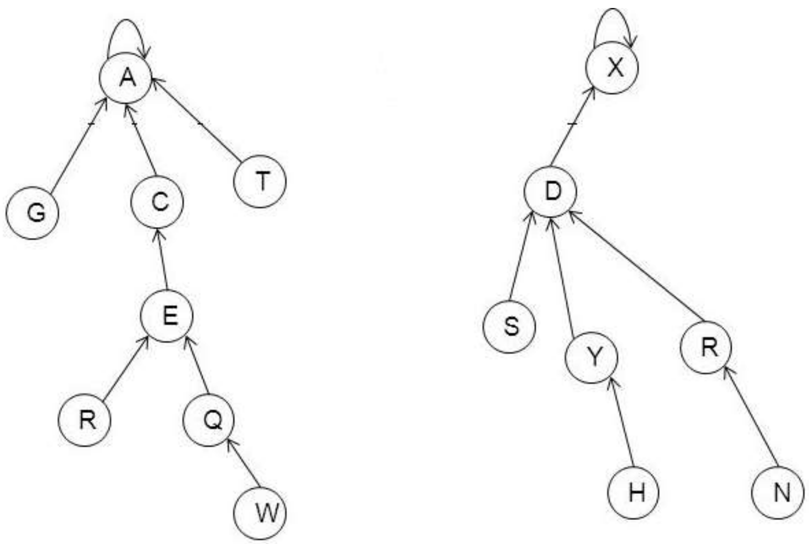
PC_FIND(x)
1. **if** $x \neq p[x]$
2. **then** $p[x] \leftarrow$ PC_FIND($p[x]$)
3. **return** $p[x]$

- Worst case running time for m MAKE-SET, UNION, FIND-SET operations is: $O(m \cdot \alpha(n))$, where $\alpha(n) \leq 4$. So nearly linear in m.
- The find operation does not change: $O(\log n)$

Exercise 1

- What would the resultant forest be after calling UNION(W, Y) on the disjoint-sets forest of the following figure?

  You must use the *union-by-rank* and the *path-compression* heuristics.



Exercise 2

- What would the resultant forest be after calling PC_Find(7) on the disjoint-sets forest of the following figure?

  You must use the *path-compression* heuristics.