Learn DAA : From B K Sharma

TCS-503: Design and Analysis of Algorithms

Unit II

Advanced Data Structures

Data Structure for Disjoint Sets

Unit II

- Advanced Data Structures:
  - Red-Black Trees
  - Augmenting Data Structure
  - B-Trees
  - Binomial Heaps
  - Fibonacci Heaps
  - Data Structure for Disjoint Sets

Learn DAA : From B K Sharma

Data Structure For Disjoint Sets

A. K. A.

Union-Find Structure

This data structure supports two important algorithms:

Union-Find Algorithms.

Data Structure For Disjoint Sets

Set : A collection of (distinguishable) elements.

Disjoint Sets: Two sets are disjoint if they have no common elements.

Dynamic sets : Sets that can grow, shrink, or otherwise change over time.

Disjoint-Set: A disjoint-set is a collection $S=\{S_1, S_2,\dots, S_k\}$ of distinct dynamic sets.

Let $S_1=\{c, h, e, b\}$ and $S_2=\{f, g, d\}$, Then

$S=\{S_1, S_2\}$

$S=\{c, h, e, b, f, g, d\}$ is a disjoint-set.

Learn DAA : From B K Sharma

Disjoint-Set Data Structures

Maintains a collection of disjoint sets.

Each set is identified by the representative element, which is some member of the set.

Choice of representative is dependent on the application –

it could be any member, or there might be certain criteria that determine which element to use.

Learn DAA : From B K Sharma

Disjoint-Set Data Structures

Supported operations:

Make-Set (x): creates a new set {x} whose only member is x.

Union (x, y): unites the dynamic sets that contain x and y.

Choose a representative for it.

Find-Set (x): Given object x, return (a pointer to) the representative of the set containing x

Learn DAA : From B K Sharma

Disjoint-Set Data Structures

Example :

MAKE-SET(1)	{1}	
MAKE-SET(2)	{2}	
MAKE-SET(3)	{3}	
MAKE-SET(4)	{4}	
FIND-Set(3)	(returns 3)	
FIND-Set(2)	(returns 2)	
UNION(1,2)	{1,2}	(representative 1, say)
FIND-Set(2)	(returns 1)	
FIND-Set(1)	(returns 1)	

UNION(3,4)	{3,4}
	(representative 4, say)
FIND-Set(3)	(returns 4)
UNION(2,3)	{1,2,3,4}
	Representative 1, say

Learn DAA : From B K Sharma

Union-Find Algorithms

Disjoint set algorithms are sometimes called *union-find* algorithms.

A union-find algorithm is an algorithm that performs two useful operations on disjoint set data structure:

> *Find*: Determine which set a particular element is in. Also useful for determining if two elements are in the same set.
>
> *Union*: Combine or merge two sets into a single set.

Because it supports these two operations, a disjoint-set data structure is sometimes called a *union-find data structure* or *merge-find* set.

Representation of Disjoint Sets

Linked- List Representation

Represent each set as a linked-list, with head and tail pointers and each node contains:

Representative Pointer
Value
Next-Node Pointer

Learn DAA : From B K Sharma

Representation of Disjoint Sets

Linked- List Representation

Analysis of Union (x,y):

MAKE-SET costs $O(1)$: just create a single element list.

FIND-SET costs $O(1)$: just return back-to-representative pointer.

Each UNION takes time linear in the y's length.

Suppose n MAKE-SET(y_i) operations ($O(1)$ each) followed by n-1 UNION:

- UNION(y_1, y_2), $O(1)$,
- UNION(y_2, y_3), $O(2)$,
- …..
- UNION(y_{n-1}, y_n), $O(n-1)$

The UNIONs cost $1+2+\ldots+n-1=\Theta(n^2)$

Learn DAA : From B K Sharma

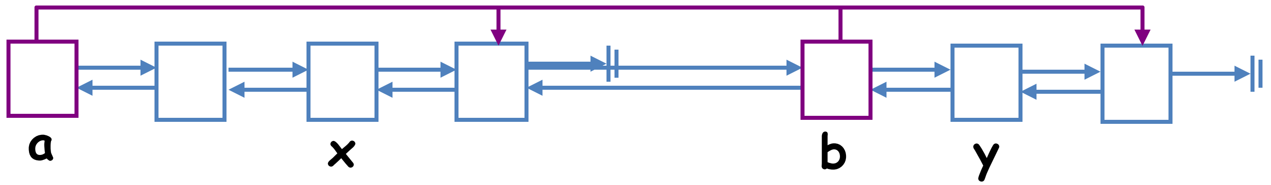
Representation of Disjoint Sets

Linked- List Representation

Union By Size:

Append the shorter list onto the longer list:

less rep pointers need to be updated.



Analysis

A sequence of m operations, of which n are Make-Set, takes Θ(m + n log n) time in the worst case.

Learn DAA : From B K Sharma

Representation of Disjoint Sets

Tree Representation

Each set is stored in a tree:

Nodes have only a pointer p[x] that points to their parent.

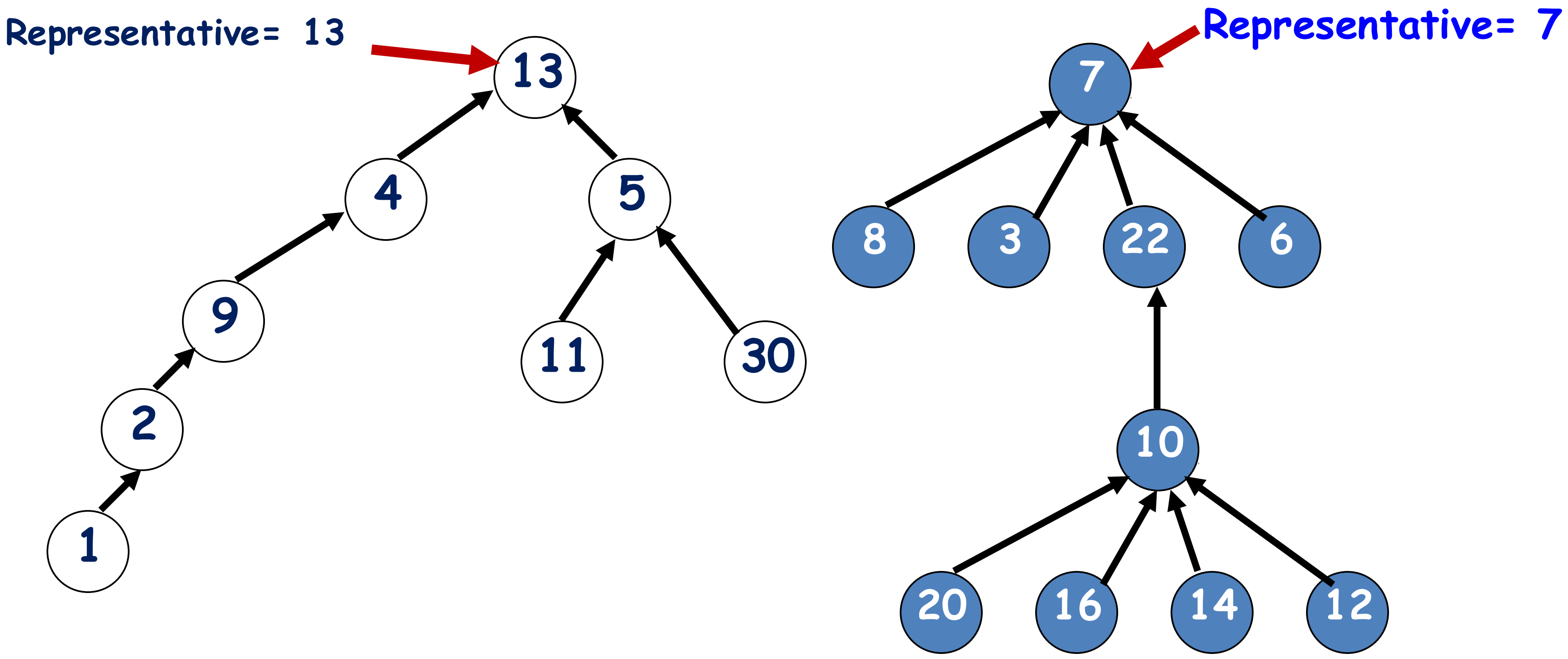
The root is the representative of the set.

The parent-pointer p[x] of the root points to the root.

Learn DAA : From B K Sharma

Representation of Disjoint Sets

Tree Representation



Representation of Disjoint Sets

Tree Representation

When joining two trees, which of the two roots should become the new root?

The best-known solution by using the following two techniques:

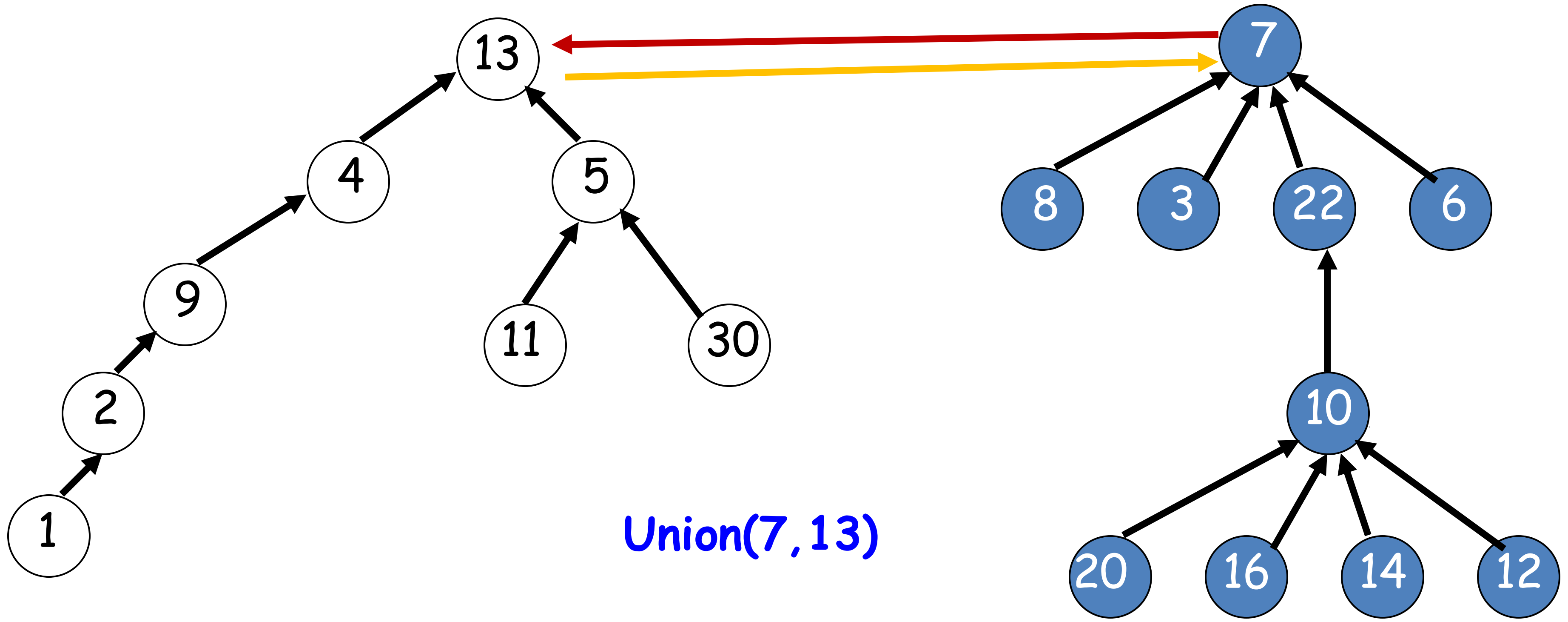
Union by rank:

Path compression:

Learn DAA : From B K Sharma

Representation of Disjoint Sets

Tree Representation: Union Operation



Which tree should become a subtree of the other???

Representation of Disjoint Sets

Tree Representation: Smart Union Approaches

There are two approaches to apply union operation:

$$\left.\begin{array}{l}\text{Union by Height}\\ \text{Union by Weight}\end{array}\right\}\ \text{a. k. a. Union by Rank}$$

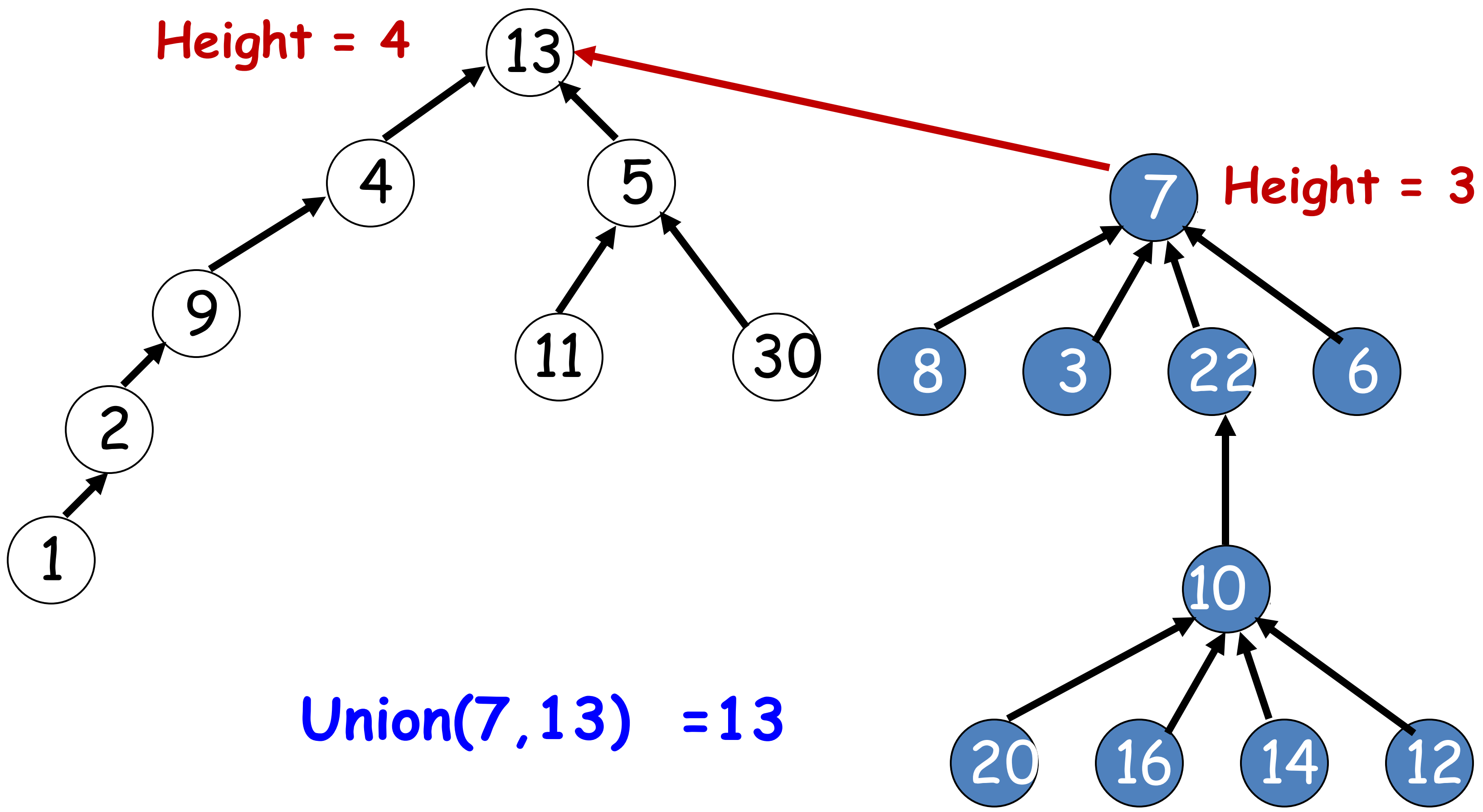
Learn DAA : From B K Sharma

Representation of Disjoint Sets

Tree Representation: Smart Union Approaches

Union by Height

Make tree with smaller height a sub-tree of the other tree.



Union(7,13) =13

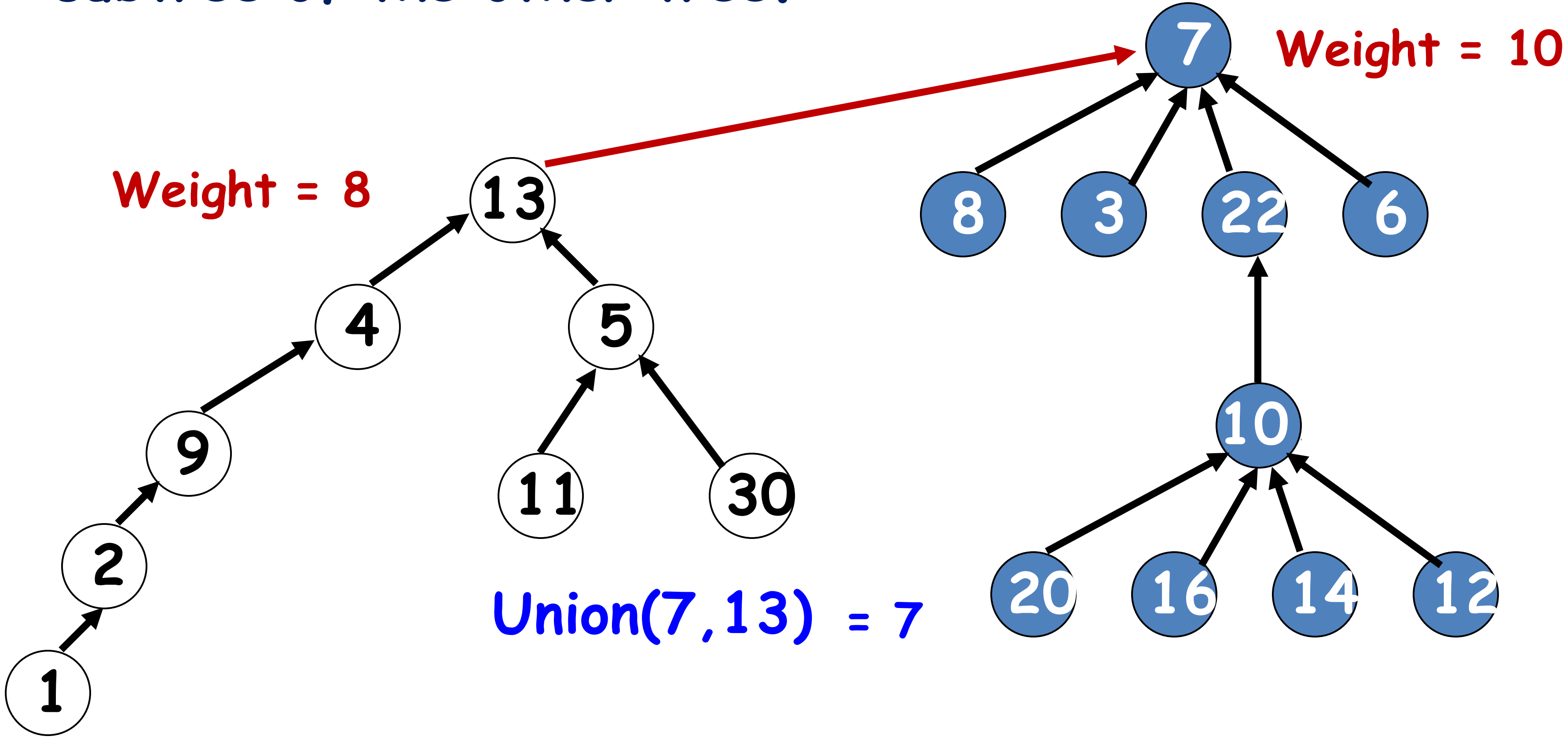
Learn DAA : From B K Sharma

Representation of Disjoint Sets

Tree Representation: Smart Union Approaches

Union by Weight

The tree with fewer number of elements becomes subtree of the other tree.



Union(7,13) = 7

Representation of Disjoint Sets

Tree Representation: Smart Find Operation

Find(x) is to identify the set that contains element x.

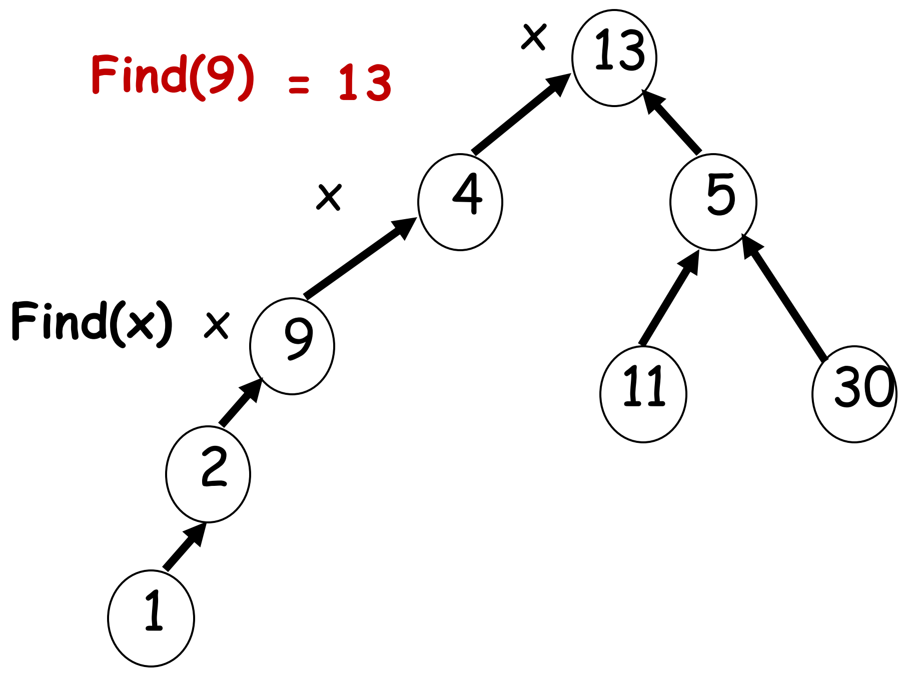
Start at the node that represents element x and climb up the tree until the root is reached.

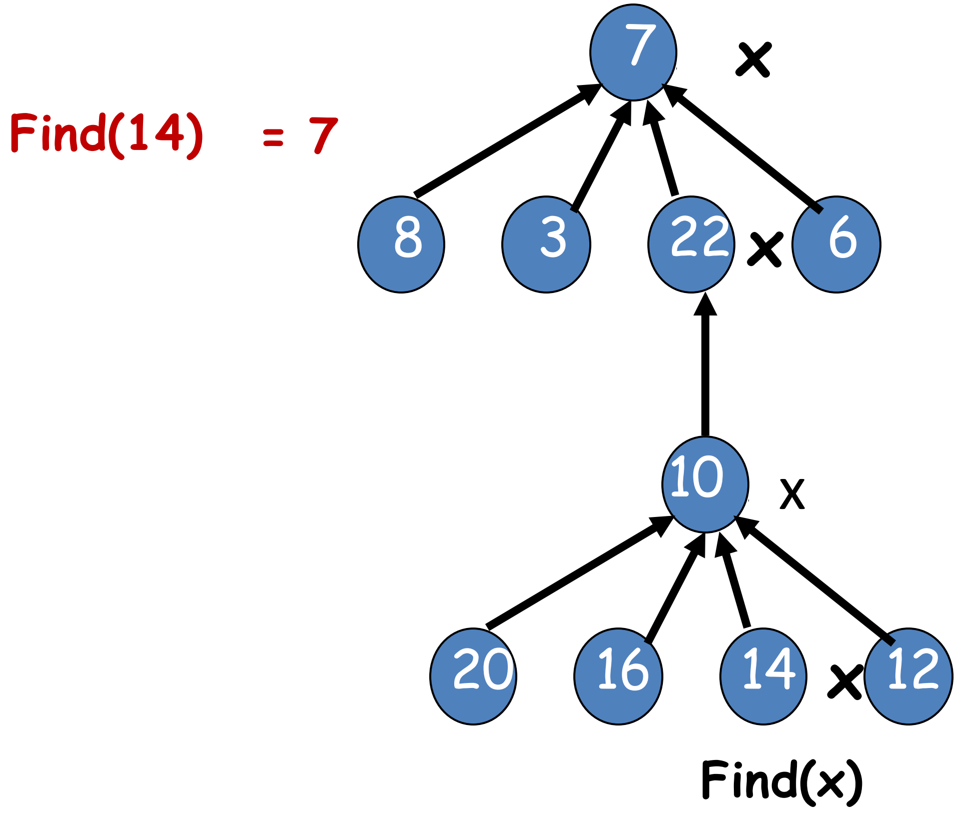
Return the element in the root (Representative).

Learn DAA : From B K Sharma

Representation of Disjoint Sets

Tree Representation: Smart Find Operation



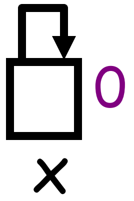


Learn DAA : From B K Sharma

Union By Rank and Path Compression

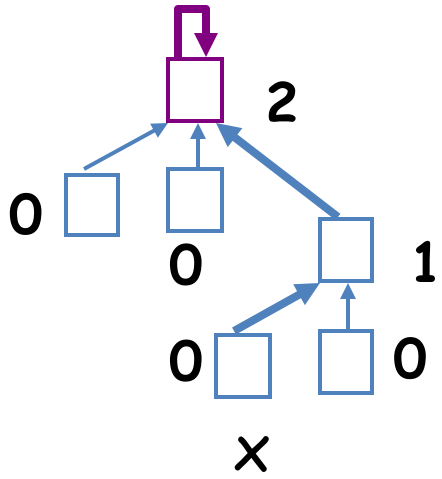
Make-Set(x)

```
1.  p[x] ← x
2.  rank[x] ← 0
```



Find-Set(x)

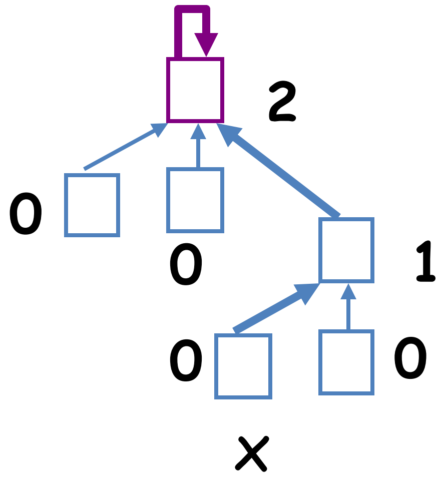
```
1.  if x ≠ p[x]
2.      p[x] ← Find-Set(p[x])
3.  return p[x]
```

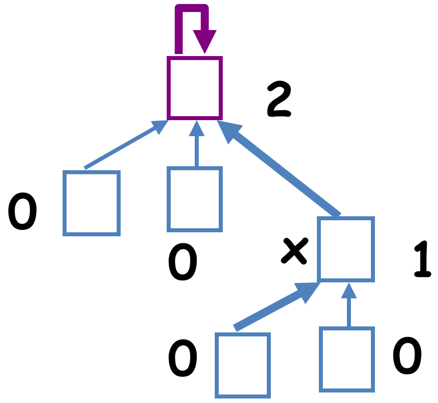


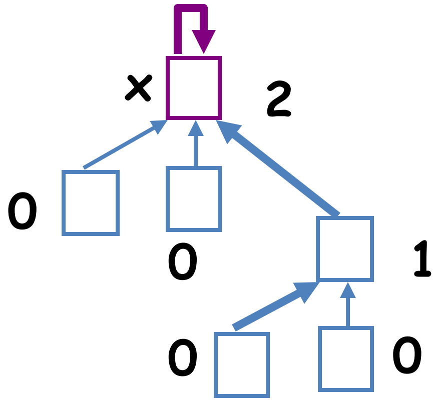
Learn DAA : From B K Sharma

Union By Rank and Path Compression

```
Find-Set(x)
1.  if x ≠ p[x]
2.      then return Find-Set(p[x])
3. else return x
```







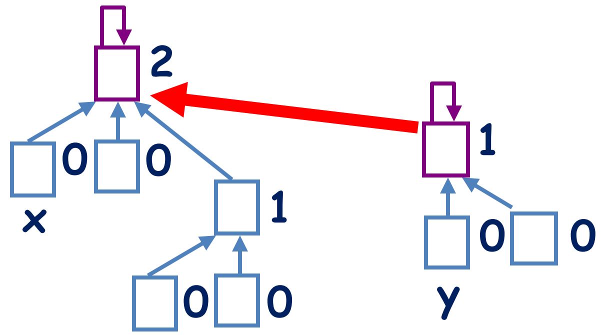
Learn DAA : From B K Sharma

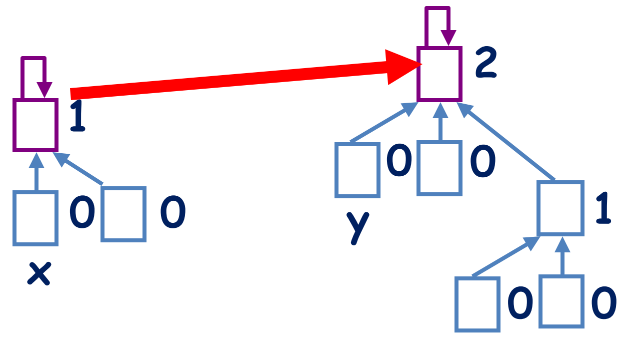
Union By Rank and Path Compression

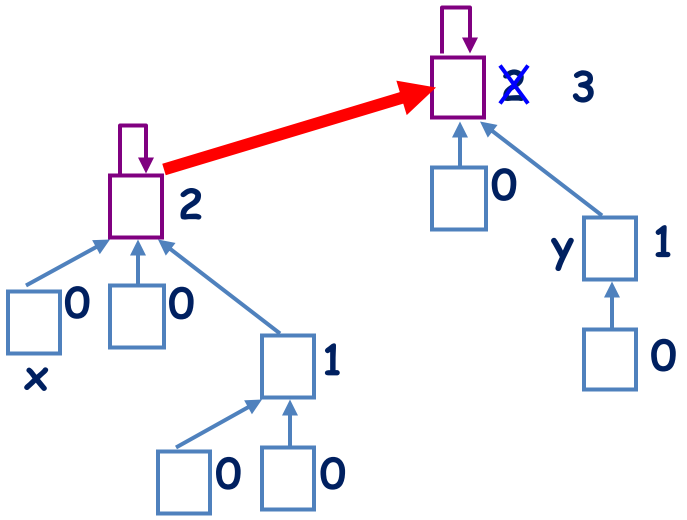
```
Union(x, y)
1.  a ← Find-Set(x); b ← Find-Set(y);
2.  if rank[a] > rank[b]
3.      then p[b] ← a
4.  else if rank[a] < rank[b]
5.      then p[a] ← b
6.  else
7.           p[a] ← b;
8.           rank[b] ← rank[b] + 1
```

Learn DAA : From B K Sharma

Union By Rank and Path Compression







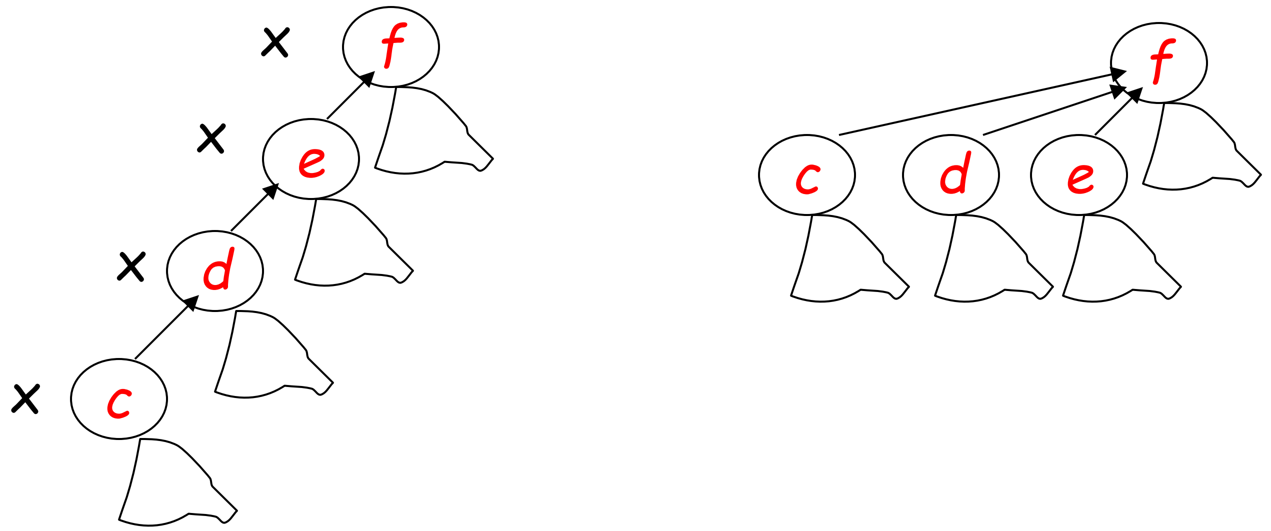
Learn DAA : From B K Sharma

Representation of Disjoint Sets

Tree Representation

Path compression:

Find-Set (c)



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.

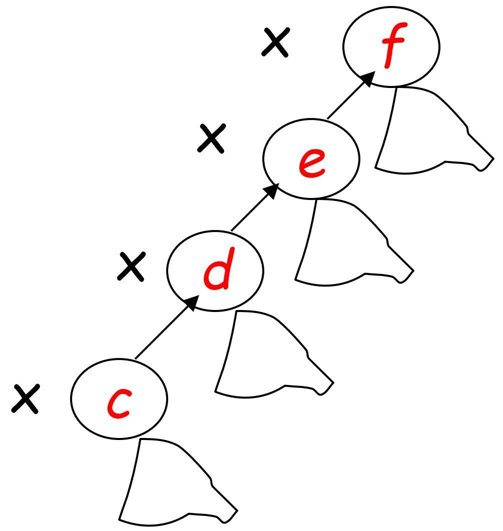
Learn DAA : From B K Sharma

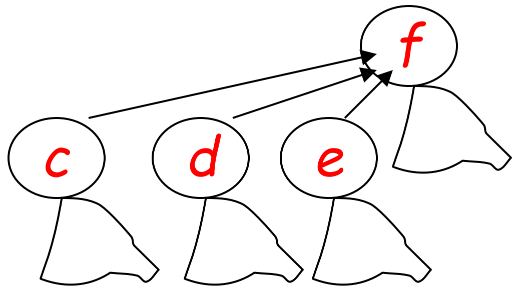
Representation of Disjoint Sets

Tree Representation

Path compression:

Find-Set (c)





Learn DAA : From B K Sharma

Representation of Disjoint Sets

Tree Representation

Path compression:

Later Find operations will have lower costs as their depths have been reduced.

Any Find operation reduces the cost of future ones.

Worst case running time for m MAKE-SET, UNION, FIND-SET operations is: $O(m\alpha(n))$ where $\alpha(n) \leq 4$.
So nearly linear in m.

END

Learn DAA : From B K Sharma

Representation of Disjoint Sets

Tree Representation

Union-Find: Merging Classes of a Partition

When joining two trees, which of the two roots should become the new root?

The best-known solution by using the following two techniques:

Union by rank:

Each node has rank field, which starts on insertion as 0.

Each time we join two classes, the root with the larger rank becomes the new root, and if both roots have the same rank, we increase the rank of one of them.

(rank is either the height or the weight of the tree)

Path compression:

After each update, we go along the path and make all the nodes point directly to the root.

Learn DAA : From B K Sharma

Representation of Disjoint Sets

Tree Representation

Union-Find: Merging Classes of a Partition

In Union we attach a smaller tree to the larger tree, results in logarithmic depth.

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