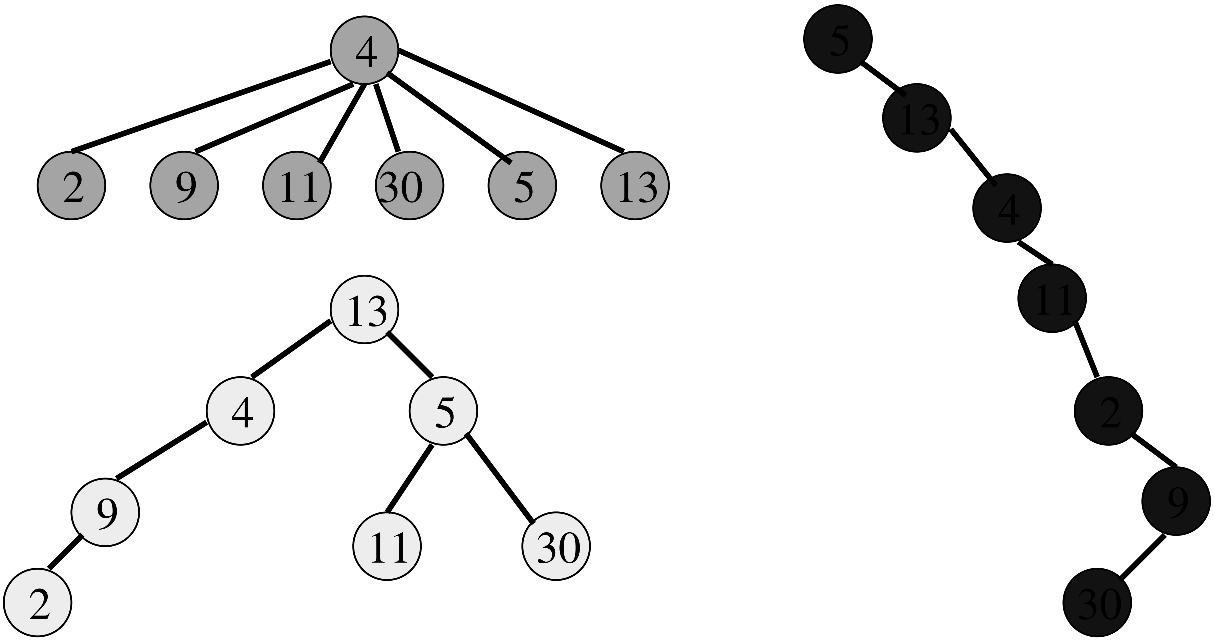
Disjoint Sets


- Given a set {1, 2, …, n} of n elements.
- Initially each element is in a different set.
  - {1}, {2}, …, {n}
- An intermixed sequence of union and find operations is performed.
- A union operation combines two sets into one.
- A find operation identifies the set that contains a particular element.

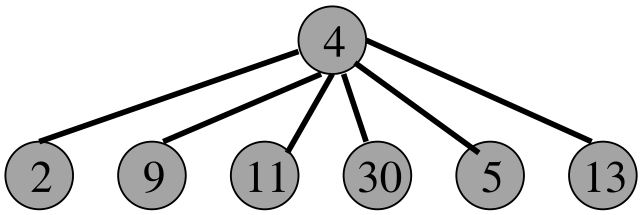
A Set As A Tree

- S = {2, 4, 5, 9, 11, 13, 30}
- Some possible tree representations:



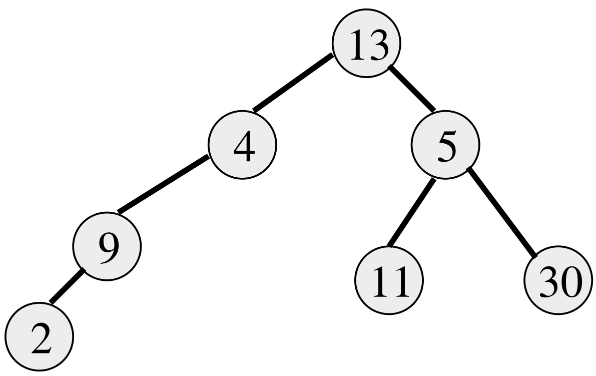
Result Of A Find Operation

- Find(i) is to identify the set that contains element i.
- In most applications of the union-find problem, the user does not provide set identifiers.
- The requirement is that Find(i) and Find(j) return the same value iff elements i and j are in the same set.



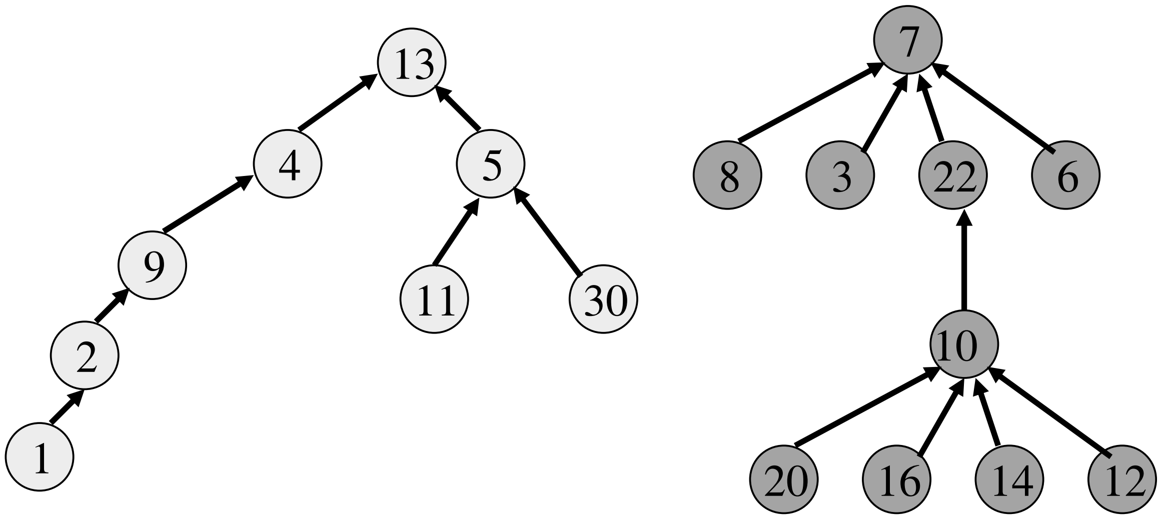
Find(i) will return the element that is in the tree root.

Strategy For Find(i)



- Start at the node that represents element i and climb up the tree until the root is reached.
- Return the element in the root.
- To climb the tree, each node must have a parent pointer.

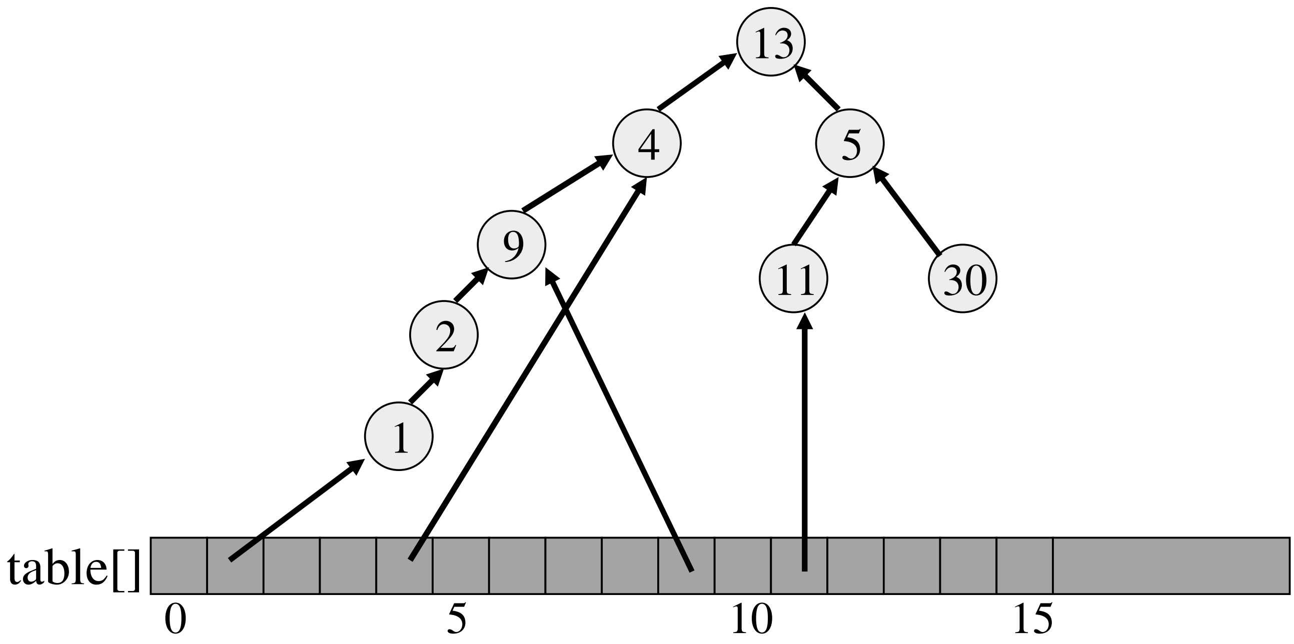
Trees With Parent Pointers



Possible Node Structure

- Use nodes that have two fields: element and parent.
  - Use an array table[] such that table[i] is a pointer to the node whose element is i.
  - To do a Find(i) operation, start at the node given by table[i] and follow parent fields until a node whose parent field is null is reached.
  - Return element in this root node.

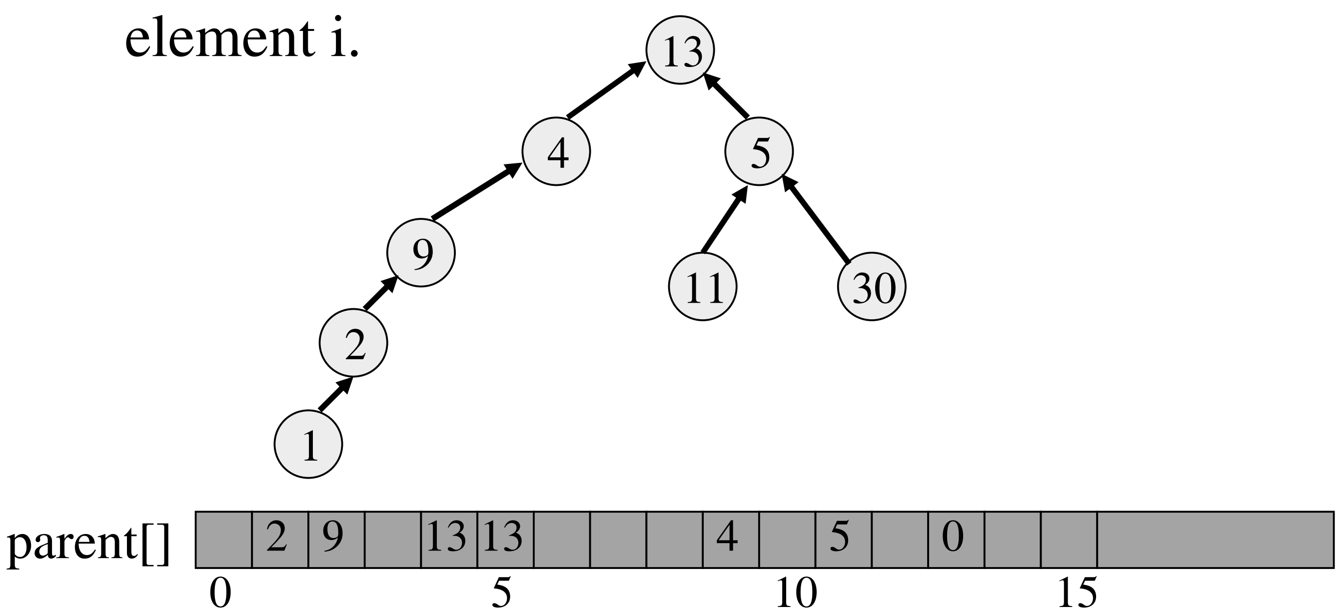
Example



(Only some table entries are shown.)

Better Representation

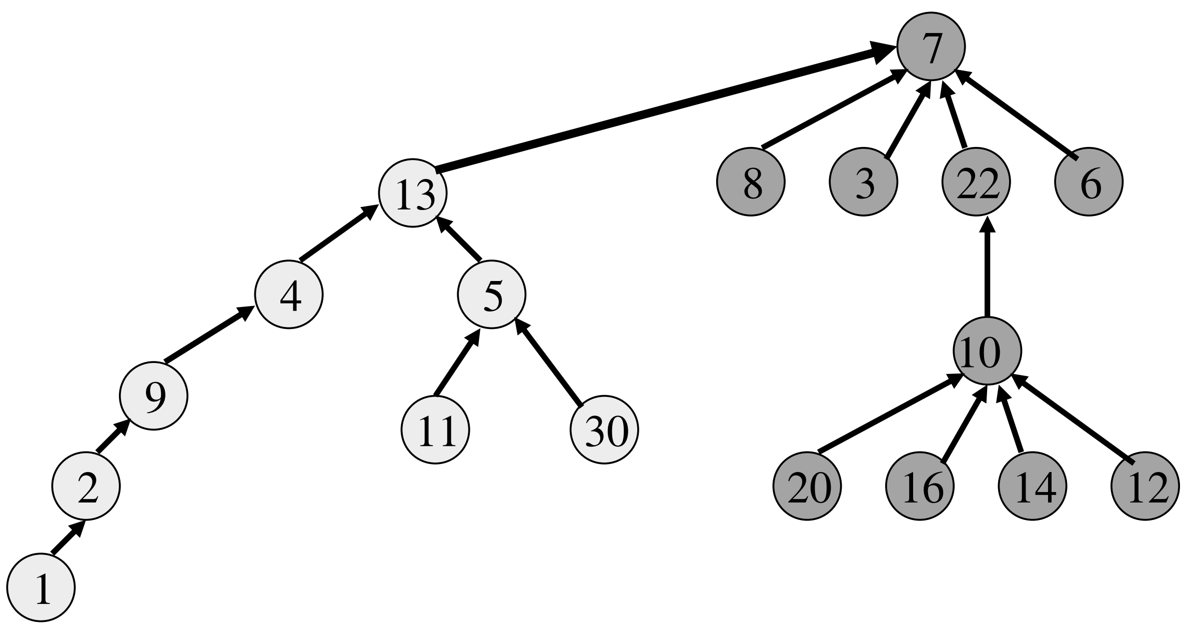
- Use an integer array parent[] such that parent[i] is the element that is the parent of element i.



Simple Union Operation

- Union(i,j)
  - i and j are the roots of two different trees, i != j.
- To unite the trees, make one tree a subtree of the other.
  - parent[j] = i

Simple Union Example



- Union(7,13)

}

Simple Union Method

```
void SimpleUnion(int i, int j)
    {parent[i] = j;}
```

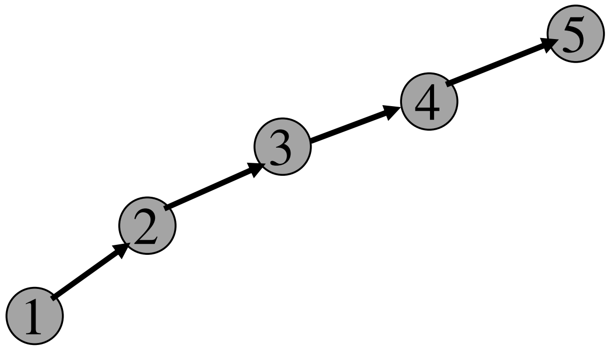
The time complexity O(1)

Simple Find Method

```
int SimpleFind(int i)
{
    while (parent[i] >= 0)
      i = parent[i];  // move up the tree
    return i;
}
```

Time Complexity of SimpleFind()

- Tree height may equal number of elements n in tree.
  - Union(2,1), Union(3,2), Union(4,3), Union(5,4)…

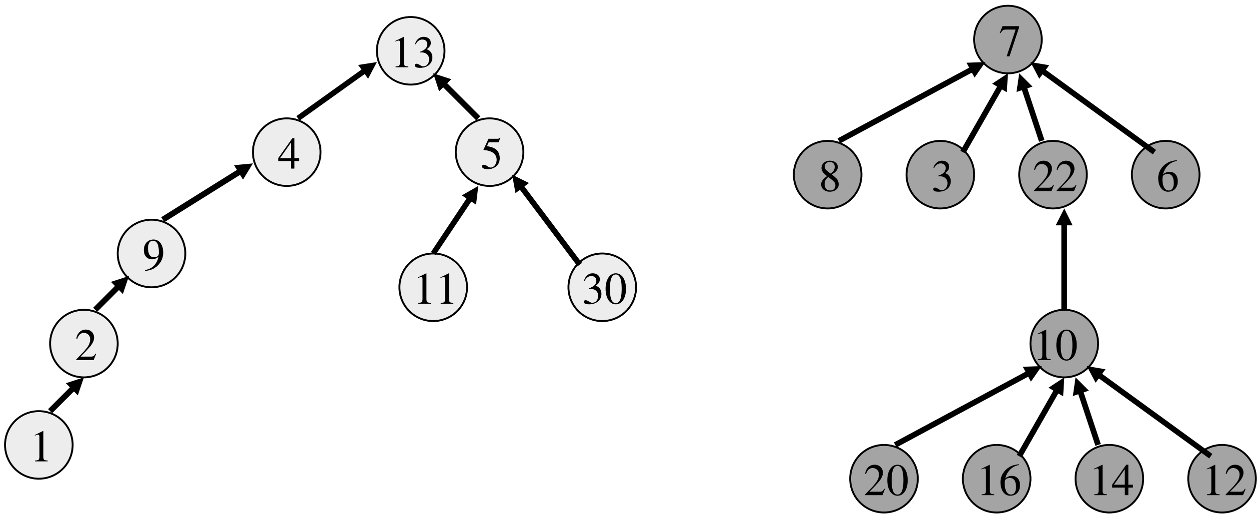


So complexity is O(n).

Time Complexity of SimpleFind()

- For a tree with height n
  – The find operation for a node at level i is O(i)
  – The total time for finding all nodes
  – O(1+2+3+…+n)=O(n^2)
  – The cost is too large.

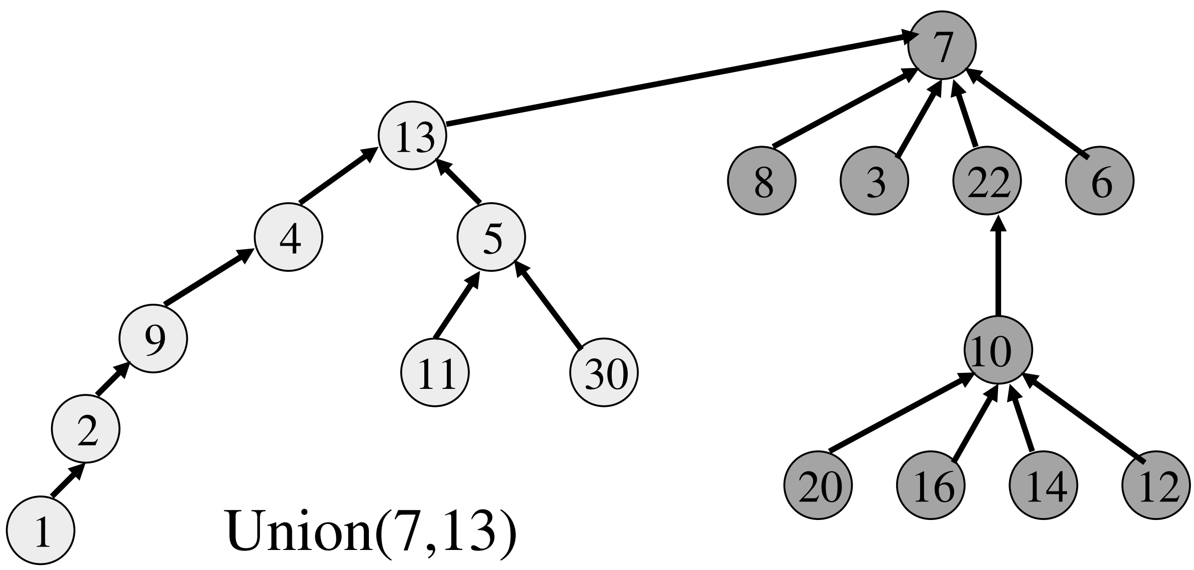
Smart Union Strategies



- Union(7,13)
- Which tree should become a subtree of the other?

Weight Rule

- Make tree with fewer number of elements a subtree of the other tree.



Implementation

- Root of each tree must record either its height or the number of elements in the tree.
- When a union is done using the height rule, the height increases only when two trees of equal height are united.
- When the weight rule is used, the weight of the new tree is the sum of the weights of the trees that are united.

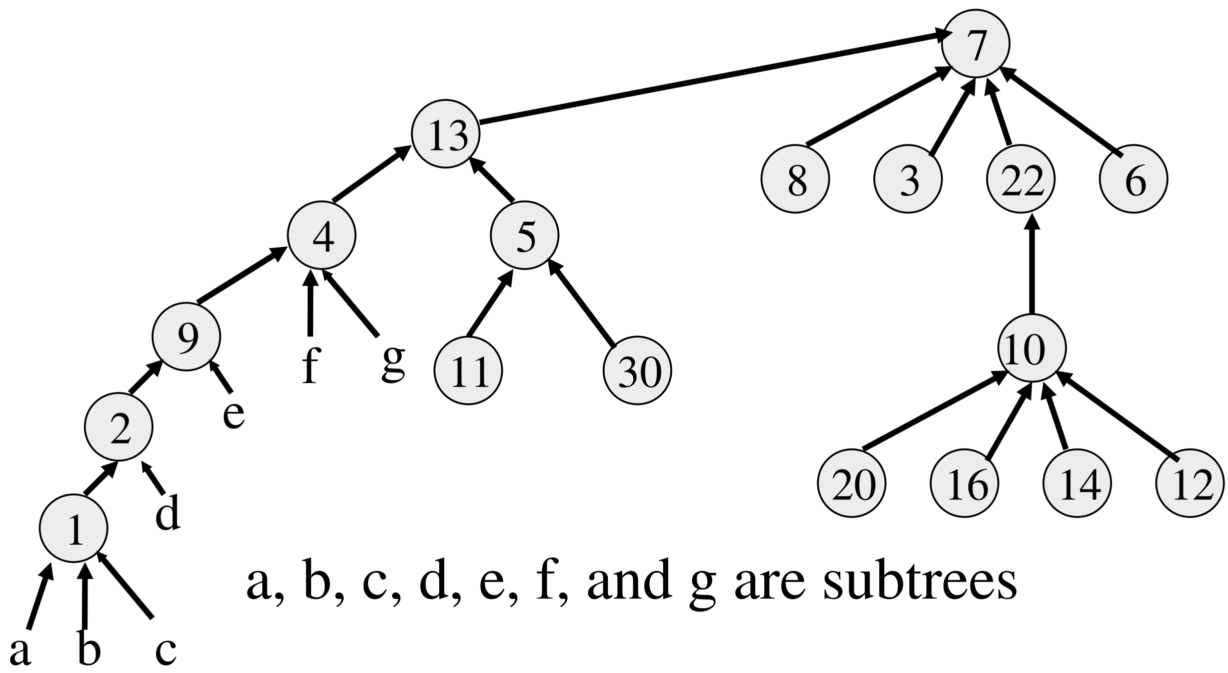
Height Of A Tree

- Suppose we start with single element trees and perform unions using either the height or the weight rule.
- The height of a tree with p elements is at most floor ($\log_2 p$) + 1.
- Proof is by induction on p. See text.

Proof

- m=1 → Clearly true
- Assume it is true for all trees with i nodes, i<=m-1 → Show that it's true for i=m
- Consider Union(k,j)
  - j has a nodes, a<=m/2
  - k has m-a nodes
  - The height is
    - the height of k: $\lfloor \log_2 (m-a) \rfloor + 1 \le \lfloor \log_2 m \rfloor + 1$
    - the height of j+1 $\lfloor \log_2 a \rfloor + 2 \le \lfloor \log_2 m/2 \rfloor + 2 \le \lfloor \log_2 m \rfloor + 1$

Sprucing Up The Find Method

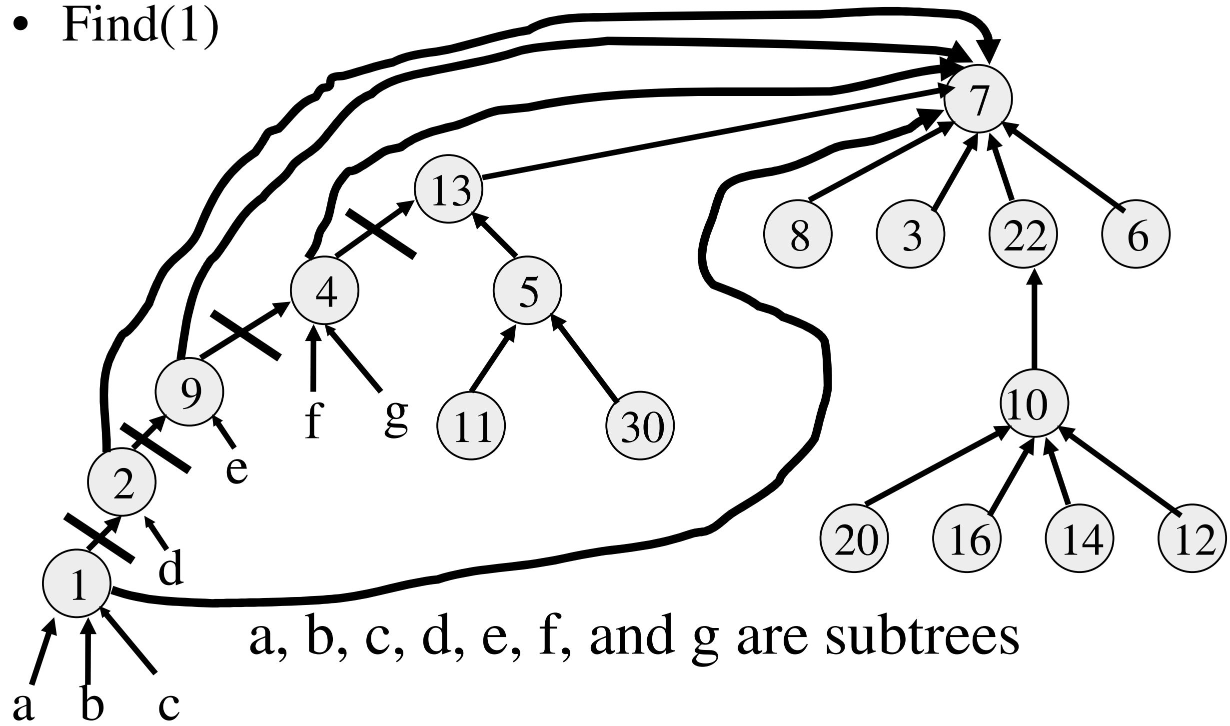


a, b, c, d, e, f, and g are subtrees

- Do Find(1) many times → It costs time to find the root
- Do additional work to make future finds easier.

Path Compaction (See Program 5.26)

- Make all nodes on find path point to tree root.
- Find(1)

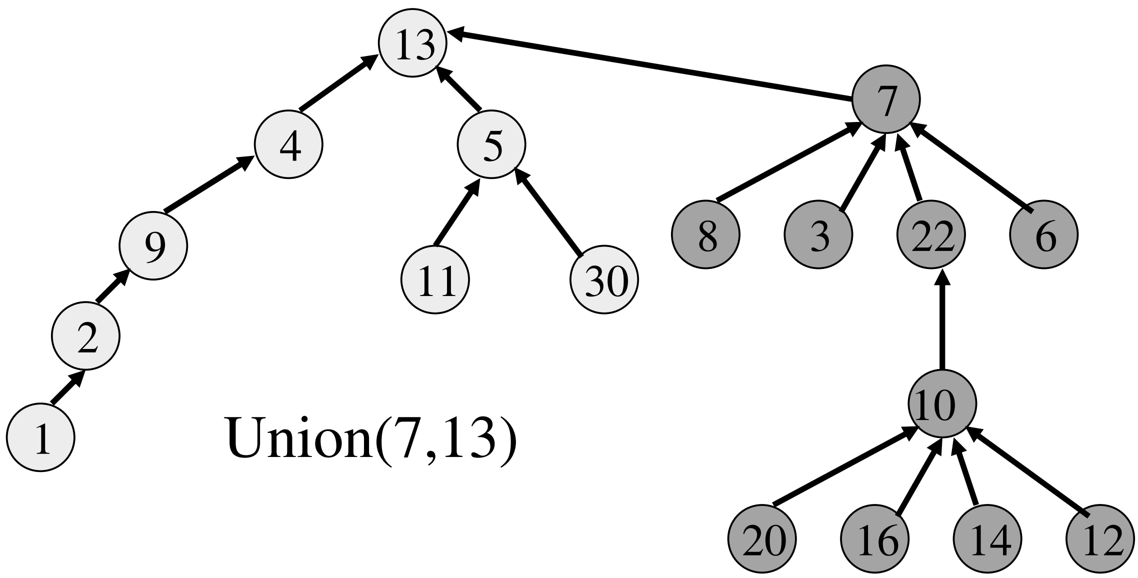


a, b, c, d, e, f, and g are subtrees

Makes two passes up the tree.

Homework: Height Rule

- Sec. 5.10 Exercise 4@P316
- Make tree with smaller height a subtree of the other tree.



Union(7,13)