Union-Find Problem

- Given a set \{1, 2, \ldots, n\} of \(n\) elements.

- Initially each element is in a different set.
  \{1\}, \{2\}, \ldots, \{n\}

- An intermixed sequence of union and find operations is performed.

- A union operation combines two sets into one.
  - Each of the \(n\) elements is in exactly one set at any time.

- FIND-SET returns the set that contains a particular element.
Set as a Tree

- $S = \{2, 4, 5, 9, 11, 13, 30\}$

Some possible tree representations:
Result of A FIND-SET Operation

- FIND-SET(i) returns the set that contains element i.
- The requirement is that FIND-SET(i) and FIND-SET(j) return the same value iff elements i and j are in the same set.

FIND-SET(i) will return the element that is in the tree root.
Strategy For FIND-SET(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
Node Structure

- Use nodes that have two fields: **element** and **parent**.
- Use an array such that `table[i]` is a pointer to the node whose element is `i`.

**Example**

```
0 4 9
```

(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`.

```
pARENT[]        0 1 2 3 4 5
                9 15
                 2 9 13 13
                  4 5 0 15
                  11 30
```

```
  13
    4
     9
    / \
   /   \
 2    5
  /  \
/    \
1    11
   / \
  30 \\
```
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
Union Example

• union(7, 13)
The FIND-SET Algorithm

```java
public int FIND-SET(int theElement)
{
    while (parent[theElement] != 0)
    {
        theElement = parent[theElement];   // move up
    }
    return theElement;
}
```
The Union Algorithm

```java
public void union(int rootA, int rootB)
    {parent[rootB] = rootA;}
```

Time Complexity of `union()` ➔ O(1)
Tree height may equal the number of elements in tree.

- union(2,1), union(3,2), union(4,3), union(5,4) …

So time complexity is $O(u)$, $u = \# \text{ of unions or elements in the set}$
Unions and FIND-SET Operations

- $O(u + uf) = O(uf)$

- Time to initialize $\text{parent}[i] = 0$ for all $i$ is $O(n)$.

- Total time is $O(n + uf)$

- We can do better!
• union(7,13)

• Which tree should become a subtree of the other?
Height Rule

- Make tree with smaller height a subtree of the other tree.
- Break ties arbitrarily.

union(7, 13)
Weight Rule

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

union(7, 13)
Implementation

- Root of each tree must record either its **height** or the **weight** (i.e., 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.
If we start with single element trees and perform unions using either the **height** or the **weight rule**. The **height** of a tree with $n$ elements is at most $\lceil \log_2 n \rceil + 1$.

Proof is by induction.

Therefore, FIND-SET(.) takes $O(\log n)$ in the worst case.
Spruce up FIND-SET() ➔ Path Compaction

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

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

Makes two passes up the tree.
Theorem [Tarjan and Van Leeuwen]

Let \( T(f,u) \) be the time required to process any intermixed sequence of \( f \) finds and \( u \) unions. Assume that \( \frac{n}{2} \leq u < n \)

\[
a(n + f*\alpha(f+n, n)) \leq T(f,u) \leq b(n + f*\alpha(f+n, n))
\]

where \( a \) and \( b \) are constants. These bounds apply when we start with singleton sets and use either the **weight** or **height rule** for **unions** and any one of the path compression methods for a **find**.

- Even though \( \alpha() \) grows very slowly, we cannot consider \( T(f,u) \) as a linear function of \( (n+f) \). But, for all practical purposes,

\[
T(f,u) = O(n+f)
\]

**In other words, for all practical purposes,**

**Time complexity of FIND-SET() = O(1)**

- **Space Complexity**: one node per element, \( O(n) \).
Ackermann’s function.

- $A(1,j) = 2^j$, $j \geq 1$
- $A(i,j) = A(i-1,2)$, $i \geq 2$ and $j = 1$
- $A(i,j) = A(i-1,A(i,j-1))$, $i, j \geq 2$

Inverse of Ackermann’s function.

- $\alpha(p,q) = \min\{z \geq 1 \mid A(z, p/q) > \log_2 q\}$, $p \geq q \geq 1$

Ackermann’s function grows very rapidly as $i$ and $j$ are increased.

- $A(2,4) = 2^{65,536}$

The inverse function grows very slowly.

- $\alpha(p,q) < 5$ until $q = 2^{A(4,1)}$
- $A(4,1) = A(3,2) = A(2, A(3,1)) = A(2, A(2,2)) = A(2,16) \gg A(2,4)$
  
- $A(2,2) = A(1,A(2,1)) = A(1,A(1,2)) = A(1,4) = 2^4 = 16$

In the analysis of the union-find problem, $q$ is the number, $n$, of elements; $p = n + f$; and $u \geq n/2$

For all practical purposes, $\alpha(p,q) < 5$