Disjoint Set ADT

**Universe:** \( U = \{x_1, x_2, \ldots, x_n\} \)

We maintain a collection \( C = \{S_1, \ldots, S_k\} \) of disjoint subsets:

- \( S_i \subseteq U \) for every \( i \), and
- \( S_i \cap S_j = \emptyset \) whenever \( i \neq j \)
- The i.d. of each set \( S_i \) is one of its elements \( x \in S_i \)

**Operations:**

- **Union** \((x, y)\) - replace \( S_x \) and \( S_y \) in \( C \) by \( S_x \cup S_y \)
- **Find-Set** \((x)\) - return the i.d. of the set containing \( x \)
- **Make-Set** \((x)\) - create the set \( S_x = \{x\} \)
Linked List Implementation

Running Time:
Make-Set: $O(1)$
Find-Set: $O(1)$
**Union**

in Linked List Implementation

Union($S_c, S_f$)

Running Time: $O(\text{length}[S_c]) + O(\text{length}(S_f))$

Can we avoid searching for last element?
Worst-Case Example

\[ S = \text{Make-Set}(x_1), \text{Make-Set}(x_2), \]
\[ \ldots \ldots \quad \text{Make-Set}(x_n), \]
\[ \text{Union}(x_1, x_2), \text{Union}(x_2, x_3), \text{Union}(x_3, x_4), \]
\[ \ldots \ldots \quad \text{Union}(x_{n-1}, x_n) \]

\[ \Rightarrow \quad 2n - 1 = O(n) \quad \text{Operations} \]

Running Time: \( \Theta(n) + \Theta(n^2) = \Theta(n^2) \)
Weighted Union Heuristic

• Each set id includes the length of the list
• In Union - append shorter list at end of longer

_Theorem:_ Performing $m > n$ operations takes $O(m + n \log n)$ time

$\Rightarrow O(\log n)$ per operation “on average”
Simple Forest Implementation

**Find-Set**(\(x\)) - follow pointers from \(x\) up to root

**Union**(\(x, y\)) - make \(x\) a child of \(y\) and return \(y\)
Worst-Case Example

\[ S = \text{Union}(x_1, x_2), \ \text{Union}(x_2, x_3), \ldots \]

\[ \ldots \ldots \ldots \text{Union}(x_{n-1}, x_n), \]

\[ \text{Find-Set}(x_1), \ \text{Find-Set}(x_2), \ldots \]

\[ \ldots \ldots \ldots \text{Find-Set}(x_n) \]

\[ \Rightarrow 2n - 1 = O(n) \quad \text{Operations} \]

**Running Time:** \( \Theta(n) + \Theta(n^2) = \Theta(n^2) \)
Weighted Union Heuristic

• Each node includes a weight field
  \text{weight} = \# \text{ elements in sub-tree rooted at node}

• \text{Find-Set}(x) \quad - \quad \text{as before} \quad \text{O} (\text{depth}(x))

• \text{Union}(x, y) \quad - \quad \text{always attach smaller tree below the root of larger tree} \quad \text{O}(1)
**Theorem:** Any $k$-node tree created using the weighted-union heuristic, has height $\leq \log k$

**Proof:** By induction on $k$

**Find-Set Running Time:** $O(\log n)$
Path Compression

Find-Set(c)
The function $\lg^* n$

$\lg^* n =$ the number of times we have to take the $\log_2$ of $n$ repeatedly to reach 1

- $\lg^* 2 = 1$
- $\lg^* 3 = \lg^* 4 = \lg^* 2^2 = 2$
- $\lg^* 16 = \lg^* 2^{2^2} = 3$
- $\lg^* 65536 = \lg^* 2^{2^{2^2}} = 4$

$\Rightarrow \ lg^* n \leq 5$ for all practical values of $n$
**Theorem (Tarjan):** If

\[ S = \text{a sequence of } O(n) \text{ Unions and Find-Set}s \]

The worst-case time for \( S \) with
- Weighted Unions, and
- Path Compressions

is \( O(n \lg^*n) \)

\[ \Rightarrow \text{The average time is } O(\lg^*n) \text{ per operation} \]
Ackerman's function

Define the function $A_i(x)$ inductively by

$$A_0(x) = x + 1$$

$$A_{i+1}(x) = A_i(A_i(A_i \ldots (x))))$$, where $A_i$ is applied $x+1$ times.

$$A_1(x) = 2x + 1$$

$$A_2(x) = A_1(A_1(A_1(\ldots x)))) > 2^{x+1}$$

$$A_3(x) = A_2(A_2(A_2(\ldots x)))) > 2^{2^{2^{\ldots(x)}}}$$

The Inverse Ackerman function is

$$\alpha(n) = \min\{k: A_k(1) > n \} < 5$$
Theorem (Tarjan): Let

\( S = \text{sequence of } \Omega(n) \text{ Unions and Find-Sets} \)

The worst-case time for \( S \) with
- Weighted Unions, and
- Path Compressions

is \( O(n\alpha(n)) \)

\( \Rightarrow \) The average time is \( O(\alpha(n)) \) steps per operation
Connected Components using Union-Find

Reminder:

• Every node \( v \) is connected to itself
• if \( u \) and \( v \) are in the same connected component then \( v \) is connected to \( u \) and \( u \) is connected to \( v \)

• Connected components form a partition of the nodes and so are disjoint:

\[ C_u \cap C_v \neq \emptyset \implies C_u = C_v \]
Connected-Components

Connected-Components(G)
1  for each vertex  \( v \in V[G] \)
2    Make-Set(\( v \))
3  for each edge  \( (u,v) \in E[G] \)
4    if  \( \text{Find-Set}(u) \neq \text{Find-Set}(v) \) then
5      \( \text{Union}(\text{Find-Set}(u),\text{Find-Set}(v)) \)
Connected-Components

Same-Component\((u, v)\)

1 \ if \ \text{Find-Set}\(u\) = \text{Find-Set}\(v\)
2 \quad \text{return} \quad \text{TRUE}
3 \quad \text{else}
4 \quad \quad \text{return} \quad \text{FALSE}