Equivalence Relation:
- Given prime $p$, $a \equiv b \mod p$
- Given graph $G$, vertices $u, v$,
  $u \equiv v$ if in same connected component

Union-Find:
- Given set $S = \{1, 2, \ldots, n\}$ maintain a partition supporting ops:
  - $\text{Init}(x)$: Each element in its own set $\{1, 2, \ldots, n\}$
  - $\text{Union}(S, S')$: Merge two sets $S, S'$ and replace with their union
  - $\text{Find}(x)$: Return the set containing $x$

Example:
- Suppose $S = \{1, 5, 3\}, \{2, 6, 8\}, \{3, 4, 7\}$
- $\text{Find}(S) \to S_1$, $\text{Find}(8) \to S_2$

Inverted-Tree Approach:
- Store elements of each set in tree with links to parent
  - Root node is set identifier

Example:
- Given $S = \{1, 3, 7, 10\}, \{2, 5, 6, 8, 11\}, \{4, 9\}$
- $\text{find}(s) = 1$

Array-Based Implementation:
- $\text{parent}[1..n]$, where $\text{parent}[i]$ is parent index or $0$ if root

Examples:
- Given prime $p$, $a \equiv b \mod p$
  - Example: $p = 5$
- Given graph $G$, vertices $u, v$,
  $u \equiv v$ if in same connected component

Union-Find:
  - $\text{Init}(x)$: Each element in its own set $\{1, 2, \ldots, n\}$
  - $\text{Union}(S, S')$: Merge two sets $S, S'$ and replace with their union
  - $\text{Find}(x)$: Return the set containing $x$

Example:
  - Suppose $S = \{1, 5, 3\}, \{2, 6, 8\}, \{3, 4, 7\}$
  - $\text{Find}(S) \to S_1$, $\text{Find}(8) \to S_2$

Inverted-Tree Approach:
  - Store elements of each set in tree with links to parent
    - Root node is set identifier

Example:
  - Given $S = \{1, 3, 7, 10\}, \{2, 5, 6, 8, 11\}, \{4, 9\}$
  - $\text{find}(s) = 1$
**How to Union?**

- Just link one tree under the other
- How to maintain low heights?
- Rank: Based on height of tree. Link lower rank as child

**Disjoint Set Union-Find II**

**Simple Union-Find**

**Running Time?**

- Init: \( O(n) \) - set a parents to null + ranks to 0
- Union: \( O(1) \) - constant time
- Find: \( O(\text{tree height}) \)

**What is worst case?**

We'll show tree height = \( O(\log n) \)

**Cases:**

1. \( h' = h'' \)
2. \( h' < h'' \)
3. \( h' > h'' \) (symmetrical)

**Final tree height:**

- \( h = \frac{h' + h'' + 1}{h' + h''} \)

**Final size:**

- \( n = n' + n'' \)
- \( 2h + 2 = 2h' + 2h'' = 2h' + 2h'' = 2h' \)

**Cases:**

1. \( n < 2h' \)
2. \( n = 2h' \)
3. \( n > 2h' \)
Path Compression:
- Whenever we perform find, shortcut the links so they point directly to root.
- This does not increase running big more than constant, but can speed up later finds.

Simple Union-Find performs a sequence of $m$ Union-Finds on set of size $n$ in $O(m \log m)$ time.
$\Rightarrow$ Amortized time (average per op) is $O(\log m)$.
$\Rightarrow$ Amortized time = $O(a(m,n))$

Amortized time = $O(a(m,n))$.
[For all practical purposes, this is constant time.]

Theorem: (Tarjan 1975) After init. any seq of $m$ Union-Finds (with path compression) takes total time $O(m \alpha(m,n))$.

Example:
This is why rank + height

Disjoint Set Union-Find III

Digression: Ackerman's Function
(1926) Primitive Rec Func.

for $i,j \geq 0$
$$A(i,j) = \begin{cases} j+1 & \text{if } i = 0 \\ A(i-1,1) & \text{if } i > 0, j = 0 \\ A(i-1, A(i,j-1)) & \text{otherwise} \end{cases}$$

Looks innocent, but it's a monster!

From super big to super small
Inverse of Ackerman
$$\alpha(m,n) = \min \{ i \geq 1 | A(i, \lceil m/n \rceil) > \log m \}$$

Obs: $\alpha(m,n) \leq 4$ for any imaginable values of $m,n$ ($m \geq n$)

$\alpha(m,n)$ represents $m$ universal

Digression: Ackerman's Function

Worst case - No. Find may take $O(\log n)$ time.
$\Rightarrow$ Amortized - Yes! Huge improvement!
(But hard to prove)
Announcements: Tue 2/7
- Programming Assignment due tomorrow 11:59 pm
- Homework 1 - out soon (preliminary)
  - Due: Tue, Feb 21, start of class
    → No late submissions
  - Basic data structures
    (+ amortization)
  - Trees
  - Union-find
  - Heaps (next)

How fast?

\[ n^2 \rightarrow \sqrt{n} \]

\[ 2^n \rightarrow \lg n \]

\[ 2^n \rightarrow \lg \lg n \leq 6 \]

\[ \log^* n \]

\[ \frac{\text{# particles}}{\text{universe}} \leq \frac{1}{250} \]

\[ 2^{250} \]

\[ \frac{\text{Invers} \text{ Ackerman}}{\text{Ackerman Func}} \]