Basic Operations: makeset(x) - create new set with 1 element
find(x) - return identifier for the set x belongs to.
union(x, y) - merge the sets that x and y belong to, if they don't belong to the same set.

Arrays:
makeSet(x) - allocates array of length 1, adds to the end of array of sets
find(x) - iterates through array of sets, looking for x, returns index of set that x belongs to
union(x, y) - finds set of x and set of y, allocates array of length len(find(x)) + len(find(y)), copies values from both sets into new array. Replaces one set with the new one, removes the other set.
Data Structures for Disjoint Sets

Runtime of array implementation: $n$ items

$\text{make-set}(x): \ O(1) \quad \text{why?}$

$\text{find}(x): \ O(n) \quad \text{why?}$

$\text{union}(x,y): \ O(n) \quad \text{why?}$

Can we do better?
Data Structures for Disjoint Sets

Directed Trees:
Directed because children have pointers to parent.

Sets 8 4 3, 5 6 8 3, 8 0 E 3 might look like this:

```
  A°  D'  C'
 /    /    /
B°  F°  E°
```

Each item in a set has two data members:
- parent: pointer to parent item. If item is root, points to self.
- rank: number representing the height of the subtree this item is root for.
def make_set(x):
    parent(x) = x
    rank(x) = 0

new set with 1 item, is own root.
height is 0, because no edges.
runtime: O(1)

def find(x):
    if x ≠ parent(x):
        parent(x) = find(parent(x))
    return parent(x)

item, and all ancestors
are now directly connected
to the root.
runtime: O(1)

def union(x, y):
    rx = find(x)
    ry = find(y)

    if rx == ry: return

    if rank(rx) > rank(ry):
        parent(ry) = rx
    else:
        parent(rx) = ry
        if rank(rx) == rank(ry): rank(ry) += 1

find the sets of x and y. if the same, done.
if x's set is taller, merge by making
y's root point to x's root; no height added.
otherwise, make x's root point to y's
root. if the trees were the same
height, the resulting tree is one higher.
runtime: O(find).
Example:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>for $x$ in A-H: makeSet($x$)</td>
</tr>
<tr>
<td>2</td>
<td>union(A, E)</td>
</tr>
<tr>
<td>3</td>
<td>union(f, D)</td>
</tr>
<tr>
<td>4</td>
<td>union(E, D)</td>
</tr>
<tr>
<td>5</td>
<td>union(A, B)</td>
</tr>
</tbody>
</table>
Example:

```
for x in A-H:
    makeset(x)

union (A, E)

union (H, D)

union (E, D)

union (A, B)
```
The previous properties of ranks still mostly hold:

- **p1**: if \( x \neq \text{parent}(x) \), then \( \text{rank}(x) < \text{rank}(	ext{parent}(x)) \)

- **p2**: for any root node of rank \( k \), there are \( \geq 2^k \) nodes in the tree.

- **p3**: for \( n \) nodes total, at most \( \frac{n}{2^k} \) nodes have rank \( k \).

\[ \text{max rank} = O(\log n) \]
Math Notes

\[ \log^*(n) \equiv \text{the number of times the log operation must be applied to } n \text{ for the result to be } \leq 1. \]

e.g. \[ \log(1000) \approx 9.965, \]
\[ \log(9.965) \approx 3.322 \]
\[ \log(3.322) \approx 1.732 \]
\[ \log(1.732) \approx 0.792 \]

\[ = 4 \]

\[ \log^*(1000) = 4 \]
Data Structures for Disjoint Sets: Directed Trees, Path Compression.

Amortized Analysis of runtime of find(x), with path compression.

Divide ranks of nodes into these groups:

\[ \{13, 23, 33, 43, 5, 6, \ldots, 163, \ 17, 18, \ldots, 65536\}, \ \{65537, \ldots, 2^{65536}\} \]

During the find method, we classify the edges traversed into 2 categories:

1- edges from node to another node in the same log* group.

2- edges from node to node in a higher log* group.

Enter pretend world:
If we ignore category 1, how many edges would we traverse?

What would the runtime of find() be?

What would the runtime be of an algorithm with \( n \) nodes, and \( m \) calls to find()?

Leave pretend world:
Amortized analysis of runtime of find(X), with path compression.

Consider edges of type 1.

What is the maximum number of times that a node in the \( \log^* \) group ending in \( 2^k \) will use such an edge, over the lifetime of the algorithm?

What is an easy upper limit on the number of nodes in any \( \log^* \) group?

Can you find a lower, upper limit?

Given the number of nodes in the \( \log^* \) group, and the maximum number of edges within the group per node, how many in-group edges will there be?

How many groups are there?

During the algorithm, what is the maximum number of these edges?
Number of times a node uses in-group edges:
Each time, the rank of the parent increases, and there are $O(2^k)$ ranks, $\Rightarrow O(2^k)$ in-group edges.
Easy upper limit on number in group: $O(n)$

Lower, upper limit:

$$ \leq \frac{n}{2^{k+1}} \text{ with rank } k+1 $$

$$ \leq \frac{n}{2^{k+2}} \text{ with rank } k+2 $$

$$ \vdots $$

$$ \leq \frac{n}{2^k} \text{ with rank } 2^k $$

Total for group $\leq \frac{n}{2^{k+1}} + \frac{n}{2^{k+2}} + \cdots + \frac{n}{2^k}$

Looking up the bound for this summation: $\leq \frac{n}{2^k}$

$\Rightarrow O\left(\frac{n}{2^k}\right)$
How many in-group edges across the entire group?

# per node = \( \mathcal{O}(2^k) \)

# of nodes = \( \mathcal{O}(n^{1/k}) \)

Total per group = \( \left( \frac{n}{k} \right) \cdot 2^k = \mathcal{O}(n) \)
How many in-group edges across all groups?

$\log^*(n)$ groups

$O(n)$ per group

$= 0(n \log^*(n))$
Data Structures for Disjoint Sets: Directed Trees with Path Compression

Amortized analysis of \texttt{find}(x) with path compression.

Pulling it together:

\begin{align*}
m \text{ calls to find cause } & \mathcal{O}(m \log^*(n)) \text{ between-group traversals.} \\
& \text{ and } \mathcal{O}(n \log^*(n)) \text{ in-group traversals.}
\end{align*}

Total run time is \( \mathcal{O}((n+m) \log^*(n)) \)

This is better than \( \mathcal{O}(mn \log(n)) \)!