Data Structures for Disjoint Sets

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 \(\text{len}(\text{find}(x)) + \text{len}(\text{find}(y))\), 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): \ \mathcal{O}(1) \quad \text{why?}

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

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

Can we do better?
Data Structures for Disjoint Sets

Directed Trees:

Directed because children have pointers to parent.

Sets $A^3, BDBF^3, DCE^3$ might look like this:

![Tree Diagram]

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.
Data Structures for Disjoint Sets: Directed Trees

def make_set(x):
    parent(x) = x
    rank(x) = 0

def find(x):
    while parent(x) != x:
        x = parent(x)
    return x

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

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

the parent pointer is used as the unique set identifier.
runtime: O(rank(root))

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).
# Data Structures for Disjoint Sets: Directed Trees

**Example:**

```python
def make_set(x):
    for x in 'A-H':
        make_set(x)
```

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td><img src="image.png" alt="Graph" /></td>
</tr>
<tr>
<td><strong>B</strong></td>
<td><img src="image.png" alt="Graph" /></td>
</tr>
<tr>
<td><strong>C</strong></td>
<td><img src="image.png" alt="Graph" /></td>
</tr>
<tr>
<td><strong>D</strong></td>
<td><img src="image.png" alt="Graph" /></td>
</tr>
<tr>
<td><strong>E</strong></td>
<td><img src="image.png" alt="Graph" /></td>
</tr>
<tr>
<td><strong>F</strong></td>
<td><img src="image.png" alt="Graph" /></td>
</tr>
<tr>
<td><strong>G</strong></td>
<td><img src="image.png" alt="Graph" /></td>
</tr>
<tr>
<td><strong>H</strong></td>
<td><img src="image.png" alt="Graph" /></td>
</tr>
</tbody>
</table>

**Union (A, E)**

- ![Graph](image.png)

**Union (E, F)**

- ![Graph](image.png)

**Union (G, H)**

- ![Graph](image.png)

**Union (B, E)**

- ![Graph](image.png)

**Union (E, H)**

- ![Graph](image.png)
Both \text{union}(x,y) and \text{find}(x) have runtimes that depend on the height of the trees. What is the worst case height?

- **P1:** if \( x \neq \text{parent}(x) \): \( \text{rank}(x) < \text{rank}(\text{parent}(x)) \)
makeset \((x)\) initializes this property, and \text{union}(x,y) maintains it

- **P2:** any node of rank \( k \), has \( \geq 2^k \) nodes in its subtree.
  
  \[
  \begin{align*}
  k=0, & \quad 2^0 = 1 \\
  k=1, & \quad 2^1 = 2, \text{ merge 2 trees of size } \geq 1 \\
  k=2, & \quad 2^2 = 4, \text{ merge 2 trees of size } \geq 2 \\
  k, & \quad 2^k, \text{ merge 2 trees of size } \geq 2^{k-1} \geq 2 \cdot 2^{k-1} = 2^k
  \end{align*}
  \]

- **P3:** if there are \( n \) items total, there can be at most \( \frac{n}{2^k} \) items with rank \( k \).
  
  \[
  \begin{align*}
  k=0, & \quad \frac{n}{2^0} = n \\
  k=1, & \quad \frac{n}{2^1} = \frac{n}{2} \\
  k=2, & \quad \frac{n}{2^2} = \frac{n}{4} \\
  k=k & \quad \frac{n}{2^k} = \frac{n}{2^k}
  \end{align*}
  \]

max value of \( k \)?

\[
1 = \frac{n}{2^k} \Rightarrow 2^k = n \Rightarrow k = \log_2 n \Rightarrow k_{\text{max}} = O(\log n).
\]
Data Structures for Disjoint Sets: Directed Trees

The runtime of this data structure is:

\[ \text{make set}(x) : \quad O(1) \]

\[ \text{find}(x) : \quad O(\log n) \]

\[ \text{union}(x, y) : \quad O(\log n) \]

Can we do better?