Disjoint Sets

Let $R$ be an equivalence relation. We represent $R$ as disjoint sets
- Each element exists in exactly one set.
- Every set is an equitant representation.
  - Mathematically: $4 \in [o]_R \rightarrow 8 \in [o]_R$
  - Programmatically: $\text{find}(4) == \text{find}(8)$

Building Disjoint Sets:
- Maintain a collection $S = \{s_0, s_1, \ldots s_k\}$
- Each set has a representative member

**Implementation #2:**
- Continue to use an array where the index is the key
- The value of the array is:
  - $-1$, if we have found the representative element
  - The index of the parent, if we haven’t found the rep. element

---

**Operation:** $\text{find}(k)$

**Operation:** $\text{union}(k_1, k_2)$

**Implementation #2:**
- Continue to use an array where the index is the key
- The value of the array is:
  - $-1$, if we have found the representative element
  - The index of the parent, if we haven’t found the rep. element

---

What is the running time of $\text{find}$?

What is the ideal UpTree?

How do we want to union the two UpTrees?

Building a Smart Union Function

The implementation of this visual model is the following:
What are possible strategies to employ when building a “smart union”?

**Smart Union Strategy #1:** ________________

Idea: Keep the height of the tree as small as possible!

Metadata at Root:

After union( 4, 7 ):

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Smart Union Strategy #2:** ________________

Idea: Minimize the number of nodes that increase in height. (Observe that the tree we union have all their nodes gain in height.)

Metadata at Root:

After union( 4, 7 ):

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Smart Union Implementation:**

```cpp
void DisjointSets::unionBySize(int root1, int root2) {
    int newSize = arr_[root1] + arr_[root2];
    // If arr_[root1] is less than (more negative), it is the
    // larger set; we union the smaller set, root2, with root1.
    if ( arr_[root1] < arr_[root2] ) {
        arr_[root2] = root1;
        arr_[root1] = newSize;
    }
    // Otherwise, do the opposite:
    else {
        arr_[root1] = root2;
        arr_[root2] = newSize;
    }
}
```

**How do we improve this?**

```cpp
int DisjointSets::find(int i) {
    if ( arr_[i] < 0 ) { return i; }
    else { return _find( arr_[i] ); }
}
```

```
int DisjointSets::unionBySize(int root1, int root2) {
    int newSize = arr_[root1] + arr_[root2];
    // If arr_[root1] is less than (more negative), it is the
    // larger set; we union the smaller set, root2, with root1.
    if ( arr_[root1] < arr_[root2] ) {
        arr_[root2] = root1;
        arr_[root1] = newSize;
    }
    // Otherwise, do the opposite:
    else {
        arr_[root1] = root2;
        arr_[root2] = newSize;
    }
}
```

**Running Time:**

- Worst case running time of find(k):
- Worst case running time of union(r1, r2), given roots:
- New function: “Iterated Log”:
  \[ \log^*(n) := \]

- Overall running time:
  - A total of \( m \) union/find operation runs in:

**CS 225 – Things To Be Doing:**

1. mp_mosaics due today.
2. Exam on Friday practice on PrairieLearn now
3. Daily POTDs are ongoing!