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: $4 \in [0]_R \Rightarrow 8 \in [0]_R$
  - 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
  ```cpp
  void makeSet(const T & t);
  void union(const T & k1, const T & k2);
  T & find(const T & k);
  ```

Operation: find($k$)

Operation: 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

Building a Smart Union Function

The implementation of this visual model is the following:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>4</th>
<th>2</th>
<th>7</th>
<th>3</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>-1</td>
<td>10</td>
<td>7</td>
<td>-1</td>
</tr>
</tbody>
</table>

What is the running time of find?

What is the ideal UpTree?

How do we want to union the two UpTrees?
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):

```
  6 6 6 8 10 7 7 7 4 5
```

**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):

```
  6 6 6 8 10 7 7 7 4 5
```

**Smart Union Implementation:**

```
DisjointSets.cpp (partial)
1 void DisjointSets::unionBySize(int root1, int root2) {
2   int newSize = arr_[root1] + arr_[root2];
3   if ( arr_[root1] < arr_[root2] ) {
4     arr_[root2] = root1;
5     arr_[root1] = newSize;
6   } else {
7     arr_[root1] = root2;
8     arr_[root2] = newSize;
9   }
}
```

**How do we improve this?**

```
DisjointSets.cpp (partial)
1 int DisjointSets::find(int i) {
2   if ( arr_[i] < 0 ) { return i; }
3   else { return _find( arr_[i] ); }
4 }
```

```
DisjointSets.cpp (partial)
1 void DisjointSets::unionBySize(int root1, int root2) {
2   int newSize = arr_[root1] + arr_[root2];
3   if ( arr_[root1] < arr_[root2] ) {
4     arr_[root2] = root1;
5     arr_[root1] = newSize;
6   } else {
7     arr_[root1] = root2;
8     arr_[root2] = newSize;
9   }
10  // Otherwise, do the opposite:
11  else {
12     arr_[root1] = root2;
13     arr_[root2] = newSize;
14  }
15 }
```

**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. final project repos and teams coming soon.
3. Daily POTDs are ongoing!