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 $S = \{s_0, s_1, ... s_k\}$
- Each set has a representative member
  ```
  void makeSet(const T & t);
  void union(const T & k1, const T & k2);
  T & find(const T & k);
  ```

Operation:
- $\text{find}(k)$
- $\text{union}(k_1, k_2)$

**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

The implementation of this visual model is the following:

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

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):

<table>
<thead>
<tr>
<th>6</th>
<th>6</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>7</th>
<th>7</th>
<th>7</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
</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):

<table>
<thead>
<tr>
<th>6</th>
<th>6</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>7</th>
<th>7</th>
<th>7</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
</table>

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     }  
10 }  
```

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] is less than (more negative), it is the  
4     // larger set; we union the smaller set, root2, with root1.  
5     if ( arr_[root1] < arr_[root2] ) {  
6         arr_[root2] = root1;  
7         arr_[root1] = newSize;  
8     }  
9     else {  
10        arr_[root1] = root2;  
11        arr_[root2] = newSize;  
12     }  
13 }  
```

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_mazes due released
2. Lab Section: lab_heaps starts today!
3. Daily POTDs are ongoing!