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: find(4) == find(8)

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

\[
\text{void makeSet(const T & t);} \\
\text{void union(const T & k1, const T & k2);} \\
\text{T & find(const T & k);} \\
\]

Implementation – DisjointSets::find

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

What is the running time of find?

What is the ideal UpTree?

Implementation – DisjointSets::union

```
1 void DisjointSets::union(int r1, int r2) { 
2   
3 }
```

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):

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):

Smart Union Implementation:
How do we improve this?

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_traversals due Monday
2. final project proposals due today.
3. Daily POTDs are ongoing!