Disjoint Sets
Let \( R \) be an equivalence relation. We represent \( R \) as several disjoint sets. Two key ideas from Monday:
- 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.
- ADT:
  ```
  void makeSet(const T & t);
  void union(const T & k1, const T & k2);
  T & find(const T & k);
  ```

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

**Example:**
![Diagram of disjoint sets and operations]

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

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

What is the ideal UpTree?
Implementation – DisjointSets::union

DisjointSets.cpp (partial)

```cpp
void DisjointSets::union(int r1, int r2) {
}
```

How do we want to union the two UpTrees?

Building a Smart Union Function

The implementation of this visual model is the following:

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

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  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  4  5
```

Smart Union Implementation:

```
void DisjointSets::unionBySize(int root1, int root2) {
    int newSize = arr_[root1] + arr_[root2];
    if (arr_[root1] < arr_[root2]) {
        arr_[root2] = root1;
        arr_[root1] = newSize;
    } else {
        arr_[root1] = root2;
        arr_[root2] = newSize;
    }
}
```

CS 225 – Things To Be Doing:

1. Theory Exam 3 is on-going
2. MP6 released; Extra Credit deadline on Monday, April 9th
3. lab_heaps released today
4. Daily POTDs are ongoing!