Today’s announcements:

MP6 available, due 11/18, 11:59p.
Code challenge! 11/19, 9p, in Siebel 0224.

Let $R$ be an equivalence relation on the set of students in this room, where $(s,t) \in R$ if $s$ and $t$ have the same favorite among \{AB, TR, CC, MC, ____\}.

Notation from math: $[\_\_]_R = \{x : xR\_\_\}$

One big goal for us: Given $s$ and $t$ we want to determine if $sRt$. 
A Disjoint Sets example:

Let $R$ be an equivalence relation on the set of students in this room, where $(s,t) \in R$ if $s$ and $t$ have the same favorite among \{AB, TR, CC, MC, ____\}.

1. Find(4)
2. Find(4)==Find(8)
3. If (!(Find(7)==Find(2)) then Union(Find(7),Find(2))
Disjoint Sets ADT

We will implement a data structure in support of “Disjoint Sets”:

- Maintains a collection $S = \{s_0, s_1, \ldots, s_k\}$ of disjoint sets.
- Each set has a representative member.
- Supports functions:
  
  ```
  void MakeSet(const T & k);
  void Union(const T & k1, const T & k2);
  T & Find(const T & k);
  ```

A first data structure for Disjoint Sets:

```
0 1 4
2 7
3 5 6
```

```
0 1 2 3 4 5 6 7
0 0 2 3 0 3 3 2
```
A better data structure for Disjoint Sets: UpTrees

- if array value is -1, then we’ve found a root, o/w value is index of parent.
- x and y are in the same tree iff they are in the same set.
A Disjoint Sets example:

Let $R$ be an equivalence relation on the set of students in this room, where $(s, t) \in R$ if $s$ and $t$ have the same favorite among \{AB, FN, DJ, ZH, FB\}.

1. Find(4)
2. Find(4)==Find(8)
3. If (!(Find(7)==Find(2)) then Union(Find(7),Find(2))
A better data structure for Disjoint Sets:

```c++
int DS::Find(int i) {
    if (s[i] < 0) return i;
    else return Find(s[i]);
}
```

Running time depends on ____________.

Worst case?

What’s an ideal tree?

```c++
void DS::Union(int root1, int root2) {
    _________________________________;
}
```
something to consider…
Smart unions:

Union by height:
- Keeps overall height of tree as small as possible.

Union by size:
- Increases distance to root for fewest nodes.

Both of these schemes for Union guarantee the height of the tree is __________.
Smart unions:

```cpp
int DS::Find(int i) {
    if (s[i] < 0) return i;
    else return Find(s[i]);
}

void DS::UnionBySize(int root1, int root2) {
    int newSize = s[root1]+s[root2];
    if (isBigger(root1, root2)) {
        s[root2] = root1;
        s[root1] = newSize;
    } else {
        s[root1] = root2;
        s[root2] = newSize;
    }
}
```
Path Compression:
Path Compression:

```cpp
int DS::Find(int i) {
    if (s[i] < 0) return i;
    else return Find(s[i]);
}

void DS::UnionBySize(int root1, int root2) {
    int newSize = s[root1] + s[root2];
    if (isBigger(root1, root2)) {
        s[root2] = root1;
        s[root1] = newSize;
    }
    else {
        s[root1] = root2;
        s[root2] = newSize;
    }
}
```
Analysis:

\[ \log^* n := \begin{cases} 
0 & \text{if } n \leq 1; \\
1 + \log^*(\log n) & \text{if } n > 1 
\end{cases} \]

Example:

\[2^{65536}\]

Relevant result:

In an upTree implementation of Disjoint Sets using smart union and path compression upon find...

any sequence of \(m\) union and find operations results in worst case running time of \(O(\_\_\_\_\_\_\_\_\_\_\_\_)\), where \(n\) is the number of items.

http://research.cs.vt.edu/AVresearch/UF/