E pluribus unum (Out of many, one)

— Official motto of the United States of America

B Data Structures for Disjoint Sets

In this lecture, we describe some methods for maintaining a collection of disjoint sets. Each set is represented as a pointer-based data structure, with one node per element. Each set has a 'leader' element, which uniquely identifies the set. (Since the sets are always disjoint, the same object cannot be the leader of more than one set.) We want to support the following operations.

- MAKESET(x): Create a new set $\{x\}$ containing the single element x. The element x must not appear in any other set in our collection. The leader of the new set is obviously x.
- FIND(x): Find (the leader of) the set containing x.
- UNION(A, B): Replace two sets A and B in our collection with their union $A \cup B$. For example, UNION(A, MAKESET(x)) adds a new element x to an existing set A. The sets A and B are specified by arbitrary elements, so UNION(x, y) has exactly the same behavior as UNION(FIND(x), FIND(y)).

Disjoint set data structures have lots of applications. For instance, Kruskal's minimum spanning tree algorithm relies on such a data structure to maintain the components of the intermediate spanning forest. Another application might be maintaining the connected components of a graph as new vertices and edges are added. In both these applications, we can use a disjoint-set data structure, where we keep a set for each connected component, containing that component's vertices.

B.1 Reversed Trees

One of the easiest ways to store sets is using trees. Each object points to another object, called its *parent*, except for the leader of each set, which points to itself and thus is the root of the tree. MAKESET is trivial. FIND traverses the parent pointers up to the leader. UNION just redirects the parent pointer of one leader to the other. Notice that unlike most tree data structures, objects do *not* have pointers down to their children.

```
MAKESET(x):
  parent(x) ← x
```

```
FIND(x):
  while x ≠ parent(x)
      x ← parent(x)
  return x
```

```
UNION(x, y):
  $\overline{x}$ ← FIND(x)
  $\overline{y}$ ← FIND(y)
  parent($\overline{y}$) ← $\overline{x}$
```