Disjoint Sets
aka Union-Find Structures

For the next two algorithms we need a data structure that clusters data into multiple disjoint sets of objects -- sets that have no elements in common. We need 3 operations:

- MakeSets() takes a collection of objects and turns them into sets with one element each.
- Union(set1, set2) combines two sets into one
- Find(x) returns the "name" of the set that contains element x. The name can be almost anything, as long as we know that Find(x1) == Find(x2) when x1 and x2 are in the same set, and Find(x1) != Find(x2) when x1 and x2 are in different sets.

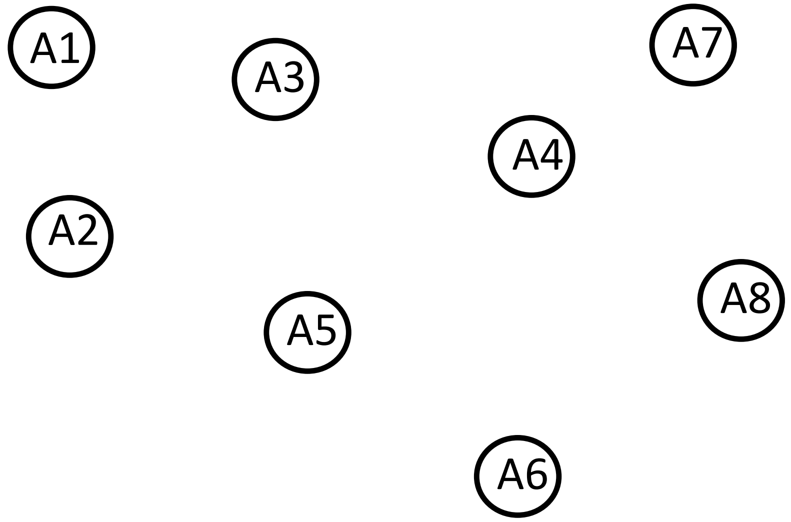
Any ideas???

Here is what we'll do: Represent elements of a set with wrappers that contain the objects, a counter, and a pointer to the set's "root". Initially each object just points to itself, and its size is 1, because all of the sets are singletons.

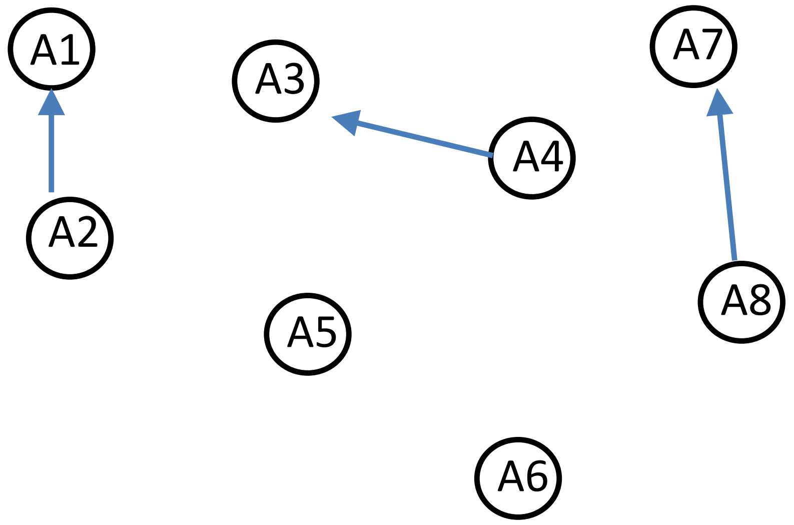
When we take the union of two sets, we make the root of the smaller one point to the root of the larger one, and adjust the size of the larger one.

The find operation returns the root of a set. We may have to do some looking for it. We start at the element and go to its root. If that node points to itself, we return it is the name of the set. If it doesn't we go to its root and see if that points to itself. This continues; eventually we will get to a node that points to itself and we return this node.

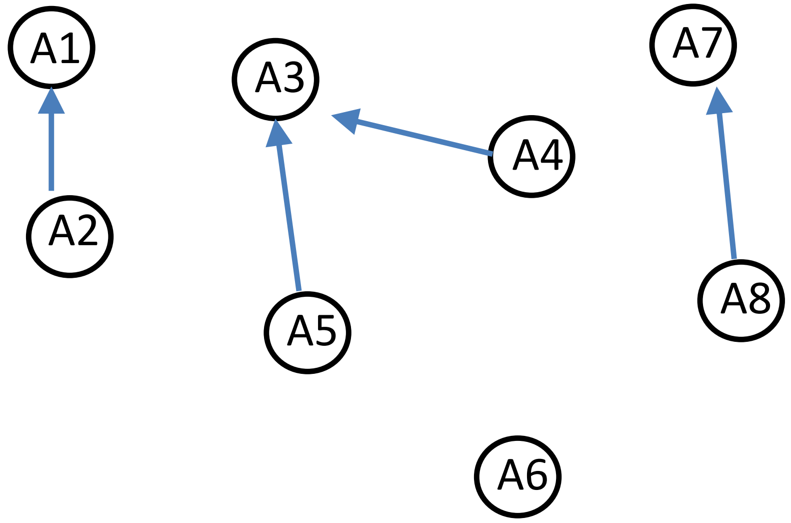
For example, suppose we have the following 8 objects:



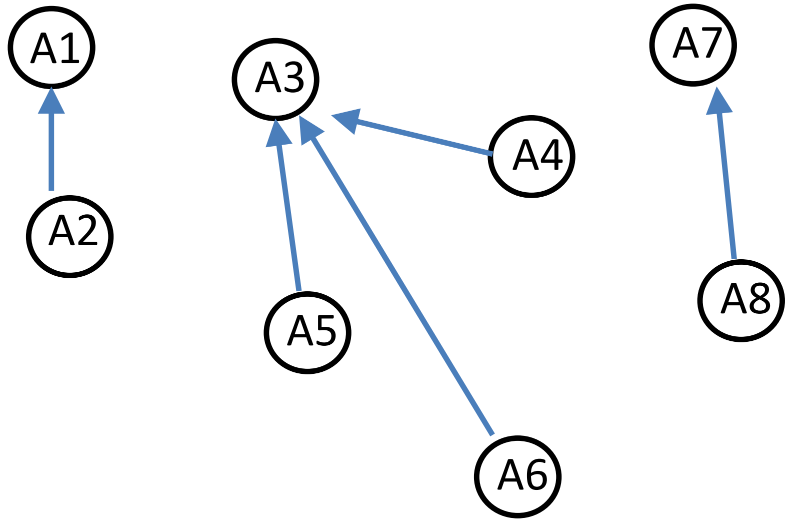
We will start by merging A1 and A2, A3 and A4, A7 and A8. Our rule is that we make the root of the smaller point at the root of the larger; these are the same size so we arbitrarily pick one:



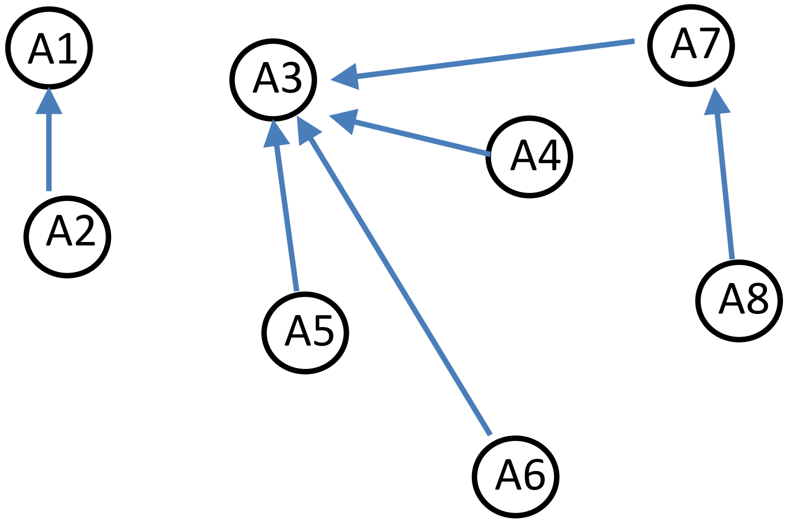
Now merge the sets containing A5 and A4. We make A5, which is its own root, point at the root of the set containing A4:



Now merge the sets containing A6 and A5:



Finally, if we merge the sets containing A5 and A8 we are left with two sets, whose roots are A1 and A3.



Now, how long does this take? Suppose we start with n objects. After we find the roots of two of these sets, forming their union is simply a matter of setting a pointer, which is constant time. Finding the root requires us to walk along a chain of pointers, with links for the previous mergers that have formed the set. How many links can there be? Each time we merge two sets, the links in the larger one stay the same and the links in the smaller one increase by 1, but the size of the set containing the increased links doubles. It can only double log(n) times before it encompasses all of the elements. So any chain of links can be at most log(n) links long.

This means that our find operation takes time O(log(n)) and our Union operation takes O(1) after the roots are found. Of course, the original MakeSets() takes time O(n), but we do this only once.