In implementing Kruskal’s algorithm we need to keep track of the components of our growing forest. This requires a data structure that represents an arbitrary collection of disjoint sets. It should support two operations:

- **find** tells one what set a value is in,
- **merge** combines two sets.

In the section on Kruskal’s algorithm, we had an array implementation that took time $O(1)$ for **find** but $O(n)$ for **merge**.

### C6.1 Disjoint Forests

Here is another idea. Again store values in an array $A$. But this time, each set is stored as a rooted sub-tree according to the following scheme:

- If $A[i] = i$, then $i$ is the label of the set and the root of some sub-tree.
- If $A[i] \neq i$, then $A[i]$ is the parent of $i$ in some sub-tree.

For example, if the components are $\{1, 2, 3, 7\}$, $\{4, 6\}$, $\{5\}$, then the array might be

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>7</td>
<td>7</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>

which represents the following sub-trees.

![Sub-trees diagram]

To determine the label of the set of a node, one follows the pointers up to the root. To combine two sets, one changes the one root to point to the root. Thus, **merge** runs in $O(1)$ time, if one is given the labels.

The **find** operation take time proportional to the depth of the sub-tree. And so we haven’t made any progress, yet. But there is a simple fix: While merging, make the root of the shorter one point to the root of the taller one. It can be shown that after a
series of $k$ merges the depth is at most $\log k$. (See exercise.) This means that the merge operation run in time $O(\log n)$. Note that one can keep track of the depth of a sub-tree. Applied to Kruskal, this gives an $O(a \log a)$ algorithm for finding a minimum spanning tree, since the initial sorting of the edges is now the most time-consuming part.

## C6.2 Pointer Jumping

There is another simple improvement to reduce the order of the running times for the data structure. Namely, one uses the work done when following the pointers to the root. Specifically: when executing find, make all pointers traversed point to the root.

It can be show that the result provides an extremely slowly growing run-time for the data structure.

### Exercises

1. Suppose in Kruskal’s algorithm we use the rooted-tree disjoint-set data structure for keeping track of components. If the nodes are A,B,C,D,E,F,G,H and the edges that Kruskal adds are in order AB, AC, DE, EF, AG, DH, GH, what does the final data structure look like:
   (a) without pointer jumping?
   (b) with pointer jumping?

2. In the rooted-tree disjoint-set data structure without pointer-jumping, show that a tree of depth $d$ has at least $2^d$ nodes.