Advanced Data Structures

Intro
Union-Find
Today
- Overview of topic
- Syllabus
- First data structure
Overview: Why?

Data structures are useful!
Often just use existing ones—
but understanding trade-offs
is key.

I'm assuming you've had an (intro)
data structures course, as well as
an algorithms course.

Reason: Beyond those "simpler"
intro ones, things get tricky.

I want to emphasize:
- Simple + elegant
- Powerful
- Useful
Next: Syllabus!
(Boring but necessary)
First data structure: Union-Find
(Have any of you already seen it?)

Goal: Keep track of a set of objects that is divided into some # of disjoint subsets, where subsets may be merged. Want to (quickly) answer queries about 2 objects being in same subset (or partition).

Why?
- Introduced in '61 by Arden, Galler & Graham, to track variables & testing equivalence. (Needed in Fortran.)
- Later: Minimum spanning trees - grow disjoint forest, until all in one tree.
Formally: 3 operations

- **makeSet(x)**: take an item to create a one element set for it
- **find(x)**: return "canonical" element of set containing x
- **union(x, y)**: Assuming that x ≠ y, form a new set that is the union of the 2 sets holding x & y, destroying the 2 old sets. (Also selects & returns a cannonical element for new set)

How to implement?

- certainly use existing DS.
Table:

Make an array/table with an entry for each element, and label with subset ID.

Ex: \( \text{makeSet}(x) \) ← 
   \( \text{makeSet}(y) \) ← 
   \( \text{makeSet}(z) \) ← 
   \( \text{union}(x, z) \) ← 
   \( \text{makeSet}(a) \) ← 
   \( \text{makeSet}(b) \) ← 
   \( \text{union}(a, x) \) ← 
   \( \text{union}(b, y) \) ← 
   \( \text{makeSet}(c) \) ←

<table>
<thead>
<tr>
<th></th>
<th>1/2</th>
<th>1/2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( I_2 \)
Runtime?

- \text{makeset} : O(1)
- \text{find} : O(1)
- \text{union} : O(n)

And: \text{O}(i)

So tradeoff w/this approach:

Bad if many unions.
Better: Use trees! (Galler + Fisher, 1964)

Each set will be a rooted tree, where elements are in the tree at the root is the canonical element.

So each element has a pointer to its parent (root points to itself)

Ex: makeset (x) makeset (y) makeset (z) union (x, z) makeset (a) makeset (b) union (a, x) union (b, y) makeset (c) union (z, b)
Then: \[\text{makeSet}(x)\]

create a node w/ value \(x\), & points its pointer to itself

\[\text{find}(x)\]:

travel up the the parent pointer of \(x\), until it points to itself

\[\text{union}(x, y)\]:

combine 2 trees into a single tree by making one of the roots a child of the other root