Chapter 3, Case Studies:

- Linear-time sorting algorithms (Monday and Wednesday)
- Disjoint sets and array forests (Today)
- Priority queues (Next week Monday)
- $N$-sets and bit vectors (Next week Wednesday)

Today:

- Problem statement
- Disjoint set ADT details
- The array forest abstraction and data structure
- Find and union strategies, with optimizations
Problem statement:

*Suppose we have a collection of items connected by an unknown equivalence relation. Efficiently find the equivalence classes in this collection as information about the relation is discovered.*
a = c
e = a + b
d = b
g = 1
f = d + c
h = e * g
The *disjoint set* ADT:

- Main operations: union two sets, find a set for a given element, and test if two elements are in the same set.
- The universe is closed.
- We assume all elements can be indexed, \([0, N)\).
- A set in the partition is identified by a leader.
Invariant (Class ArrayForestDisjointSet)

For all $i \in [0, n)$,

(a) $\text{leader}(i) = \text{leader}(\text{parents}(i))$, that is, $\text{id}(i)$ points to another element in the same set as $i$.

(b) $\text{leader}(i) = \text{parents}[\text{leader}(i)]$, that is, leaders all point to themselves.

(c) Following a finite number links implied by $\text{parents}$ will converge, that is, there is no circularity in the tree.
**DisjointSet**
- `find(int)`
- `union(int, int)`
- `connected(int, int)`
- `count()`
- `findAll(int)`

**ArrayForestDisjointSet**
- `parents: int[]`
- `finder: findStrategy`
- `unioner: unionStrategy`
- `find(int)`
- `union(int, int)`
- `connected(int, int)`
- `count()`
- `findAll(int)`

**UnionStrategy**
- `union(int, int)`

**FindStrategy**
- `find(int)`

**PlainFind**
- `finder.find(p);`

**CompressingFind**
- `finder.compress(p);`

**LazyUnion**
- `sizes: int[]`

**AggressiveUnion**

**RankingUnion**
- `sizes: int[]`

**BruteForceDisjointSet**

**Implements**
- `finder.find(p);`
- `unioner.union(p, q);`
<table>
<thead>
<tr>
<th>Union strategy</th>
<th>LazyUnion</th>
<th>AggressiveUnion</th>
<th>WeightedUnion</th>
<th>LazyUnion</th>
<th>WeightedUnion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find strategy</td>
<td>PlainFind</td>
<td>PlainFind</td>
<td>PlainFind</td>
<td>CompressingFind</td>
<td>CompressingFind</td>
</tr>
<tr>
<td>Find heavy:</td>
<td>1.30E7</td>
<td>3.34E7</td>
<td>7.40E5</td>
<td>9.26E5</td>
<td>6.68E5</td>
</tr>
<tr>
<td></td>
<td>(5.68E6)</td>
<td>(8.40E3)</td>
<td>(1.80E4)</td>
<td>(2.38E4)</td>
<td>(9.34E3)</td>
</tr>
<tr>
<td>Even mix:</td>
<td>9.89E7</td>
<td>4.41E7</td>
<td>1.20E6</td>
<td>1.56E6</td>
<td>9.80E5</td>
</tr>
<tr>
<td></td>
<td>(1.22E7)</td>
<td>(9.93E3)</td>
<td>(1.97E4)</td>
<td>(2.12E4)</td>
<td>(9.96E3)</td>
</tr>
<tr>
<td>Union heavy:</td>
<td>1.62E8</td>
<td>4.39E7</td>
<td>1.40E6</td>
<td>1.71E6</td>
<td>1.04E6</td>
</tr>
<tr>
<td></td>
<td>(1.26E7)</td>
<td>(9.99E3)</td>
<td>(2.01E4)</td>
<td>(1.59E4)</td>
<td>(1.00E4)</td>
</tr>
</tbody>
</table>


Coming up: (all end-of-day)

Do linear sorting project (suggested by Mon, Feb 7)

Due Today:
Finish reading Section 3.2 (disjoint sets and array forests)
Take disjoint-sets quiz

Due Mon, Feb 7:
Read Section 3.3 (heaps and priority queues)

Due Wed, Feb 9:
Take heap/pq quiz

Due Thurs, Feb 10:
Read Section 3.4
Do Exercises 3.(27 & 28).
Take N-sets quiz