2. Union - Find Algorithms
An illustration of the importance of algorithm design

Sedgewick, Algorithms in C, Chapter 1
Slides from
http://www.cs.princeton.edu/courses/archive/spring03/cs226/lectures/intro.4up.pdf
An Example Problem: Network Connectivity

Network connectivity.
- Nodes at grid points.
- Add connections between pairs of nodes.
- Is there a path from node A to node B?
Network Connectivity

<table>
<thead>
<tr>
<th>in</th>
<th>out</th>
<th>evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>3 4</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>4 9</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>8 0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2 3</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5 6</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>2 3 4 9</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>5 9</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>7 3</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>4 8</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>(5-6)</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>(2-3-4-8-0)</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>6 1</td>
</tr>
</tbody>
</table>
Union-Find Abstraction

What are critical operations we need to support?

- **N objects.**
  - grid points
- **FIND:** test whether two objects are in same set.
  - is there a connection between A and B?
- **UNION:** merge two sets.
  - add a connection

Design efficient data structure to store connectivity information and algorithms for UNION and FIND.

- Number of objects and operations can be huge.
Another Application: Image Processing

Find connected components.
- Read in a 2D color image and find regions of connected pixels that have the same color.
Another Application: Image Processing

Find connected components.
- Read in a 2D color image and find regions of connected pixels that have the same color.

One-pass algorithm.
- Initialize each pixel to be its own component.
- Examine pixels from left to right and top to bottom.
  - if a neighboring cell is the same color, merge current cell into same component
Objects

Elements are arbitrary objects in a network.
- Pixels in a digital photo.
- Computers in a network.
- Transistors in a computer chip.
- Web pages on the Internet.
- When programming, convenient to name them 0 to N-1.
- When drawing, fun to use animals!
Quick-Find

id[bear] = id[dragon] = id[lion] = lion
id[bat] = id[lobster] = lobster
Quick-Find Algorithm

Data structure
. Maintain array id[] with name for each component.
. If p and q are connected, then same id.
. Initialize id[i] = i.

FIND. To check if p and q are connected, check if they have the same id.

UNION. To merge components containing p and q, change all entries with id[p] to id[q].

Analysis
. FIND takes constant number of operations.
. UNION takes time proportional to N.

for (i = 0; i < N; i++)
  id[i] = 1;

if (id[p] == id[q])
  // already connected

pid = id[p];
for (i = 0; i < N; i++)
  if (id[i] == pid)
    id[i] = id[q];
Quick-Find

3-4  0 1 2 4 4 5 6 7 8 9
4-9  0 1 2 9 9 5 6 7 8 9
8-0  0 1 2 9 9 5 6 7 0 9
2-3  0 1 9 9 9 5 6 7 0 9
5-6  0 1 9 9 9 6 6 7 0 9
5-9  0 1 9 9 9 9 9 7 0 9
7-3  0 1 9 9 9 9 9 9 0 9
4-8  0 1 0 0 0 0 0 0 0 0
6-1  1 1 1 1 1 1 1 1 1 1
Problem Size and Computation Time

Rough standard for 2000.
- $10^9$ operations per second.
- $10^9$ words of main memory.
- Touch all words in approximately 1 second. (unchanged since 1950!)

Ex. Huge problem for quick find.
- $10^{10}$ edges connecting $10^9$ nodes.
- Quick-find might take $10^{20}$ operations. (10 ops per query)
- 3,000 years of computer time!

Paradoxically, quadratic algorithms get worse with newer equipment.
- New computer may be 10x as fast.
- But, has 10x as much memory so problem may be 10x bigger.
- With quadratic algorithm, takes 10x as long!
Quick-Union

Data structure: disjoint forests.
- Maintain array \( id[] \) with name for each component.
- If \( p \) and \( q \) are connected, \( p \) and \( q \) have same root, where
  - \( \text{root}(p) = id[id[...id[p]...]] \)
  - go until it doesn't change
- **FIND.** Check if \( p \) and \( q \) have same root.
- **UNION.** Set the id of \( p \)'s root to \( q \)'s root.

Analysis.
- **FIND** takes time proportional to depth of \( p \) and \( q \) in tree.
  - could be proportional to \( N \)
- **UNION** takes constant time, given roots.
Quick-Union

\[ \text{id[elephant]} = \text{skunk} \]
Quick-Union

\begin{align*}
\text{root(Tiger)} &= \text{Elephant} \\
\text{root(Lobster)} &= \text{Skunk} \\
\text{id(Elephant)} &= \text{Skunk}
\end{align*}

\text{Union(Tiger, Lobster)}
Quick-Union

3-4  0124456789
4-9  0124956789
8-0  0124956709
2-3  0194956709
5-6  0194966709
5-9  0194969709
7-3  0194969909
4-8  0194969900
6-1  1194969900
Weighted Quick-Union

Quick-find defect.
- UNION too expensive.
- Trees are flat, but too hard to keep them flat.

Quick-union defect.
- FIND could be too expensive.
- Trees could get tall.

Weighted quick-union.
- Modify quick-union to avoid tall trees.
- Keep track of size of each component.
- Balance by linking small tree below large one.
Weighted Quick-Union

Data structure: disjoint forests.
- Also maintain array \( wt[i] \) that counts the number of nodes in the tree rooted at \( i \).

FIND. Same as quick union.

UNION. Same as quick union, but:
- Merge smaller tree into the larger tree.
- Update the \( wt[] \) array.

Analysis.
- FIND takes time proportional to depth of \( p \) and \( q \) in tree.
  - depth is at most \( \lg N \)
- UNION takes constant time, given roots.

```
if (wt[i] < wt[j]) {
    id[i] = j;
    wt[j] += wt[i];
} else {
    id[j] = i;
    wt[i] += wt[j];
}
```
Weighted Quick-Union

Is performance improved?
- Theory: $\lg N$ per union or find operation.
- Practice: constant time.

Ex. Huge practical problem.
- $10^{10}$ edges connecting $10^9$ nodes.
- Reduces time from 3,000 years to 1 minute.
- Supercomputer wouldn't help much.
- Good algorithm makes solution possible.

Stop at guaranteed acceptable performance?
- Not hard to improve algorithm further.
Weighted Quick-Union with Path Compression

Path compression.
- Modify weighted quick-union to compress tree.
- Make second pass from p and q up to root, and set the id of every examined node to the new root.

```java
for (i = p; i != id[i]; i = id[i])
    id[i] = root;
for (j = q; j != id[j]; j = id[j])
    id[j] = root;
```

- No reason not to!
- In practice, keeps tree almost completely flat.
Path Compression

Find(Piggy)
Path Compression

Find(Piggy)
Weighted Quick-Union with Path Compression

**Theorem.** A sequence of \( M \) union and find operations on \( N \) elements takes \( O(N + M \lg^* N) \) time.
- Proof is difficult.
- But the algorithm is still simple!

**Remark.** \( \lg^* N \) is a constant in this universe.

**Linear algorithm?**
- Cost within constant factor of reading in the data.
- Theory: WQUPC is not quite linear.
- Practice: WQUPC is linear.

<table>
<thead>
<tr>
<th>N</th>
<th>( \lg^* N )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
</tr>
<tr>
<td>65536</td>
<td>4</td>
</tr>
<tr>
<td>2^{65536}</td>
<td>5</td>
</tr>
</tbody>
</table>
Lessons

Union-find summary.
- Online algorithm can solve problem while collecting data for "free."

"Trivial" algorithms can be useful.
- Start with simple algorithm.
  - don't use for large problems
  - can't use for huge problems
- Fast performance on test data OK.
- Strive for worst-case performance guarantees.
  - might be nontrivial to analyze
- Identify fundamental abstractions.
  - union-find
  - disjoint forests

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick-find</td>
<td>$M \cdot N$</td>
</tr>
<tr>
<td>Quick-union</td>
<td>$M \cdot N$</td>
</tr>
<tr>
<td>Weighted</td>
<td>$N + M \log N$</td>
</tr>
<tr>
<td>Path compression</td>
<td>$N + M \log N$</td>
</tr>
<tr>
<td>Weighted + path</td>
<td>$5 \cdot (M + N)$</td>
</tr>
</tbody>
</table>