Previously …

Introduction to 212
Case Study: Union-Find
makeset, find, and union operations
quick-find algorithm

Implementing a quick union

Data Structure
integer array p of length n
p[i] is the parent of i in a tree

Quick-union example

After 9 calls to makeset(…)

Quick-union example

union(1, 0)

Quick-union example

union(5, 8)

Quick-union example

union(3, 7)

Quick-union example

union(4, 6)
Implementing a quick union

How to implement \texttt{makeset(i)}?
- trivial, \( p[i] = i \)

How to implement \texttt{find(i)}?
- return root of tree containing \( i \)

How to implement \texttt{union(a, b)}?
- set \( p[\text{root of } a] \) to root of \( b \), or vice-versa

Computational Cost

<table>
<thead>
<tr>
<th></th>
<th>makeset</th>
<th>find</th>
<th>union</th>
</tr>
</thead>
<tbody>
<tr>
<td>quick-find</td>
<td>1</td>
<td>1</td>
<td>( n )</td>
</tr>
<tr>
<td>quick-union</td>
<td>1</td>
<td>( n )</td>
<td>( n )</td>
</tr>
</tbody>
</table>

If trees are tall, both \texttt{union} and \texttt{find} are expensive!

Wait … you said ‘quick union’

Need improvements …

- weighted quick-union
- path compression

Implementing a weighted quick-union

\textbf{Always} link smaller trees to larger trees

will need an extra array to keep track of \texttt{size}
Logarithmic?

When `find` traverses from a node `x` to the root each time we follow an arrow to a subtree of size at least double the size of the previous subtree.

How many arrows do we need to follow (at most) in the worst case?

\[ \log_2 n \]

Computational Cost

<table>
<thead>
<tr>
<th></th>
<th>makeSet</th>
<th>find</th>
<th>union</th>
</tr>
</thead>
<tbody>
<tr>
<td>quick-find</td>
<td>1</td>
<td>1</td>
<td>n</td>
</tr>
<tr>
<td>quick-union</td>
<td>1</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>weighted-quick-union</td>
<td>1</td>
<td>log n</td>
<td>log n</td>
</tr>
</tbody>
</table>

Processing `m` union operations takes `m \cdot \log_2 n` accesses (`m` calls to `union`).

If `m \approx n`, linearithmic time (\(n \log_2 n\) accesses)

Is linearithmic running time any good?

<table>
<thead>
<tr>
<th>Size of Input</th>
<th>(N^2)</th>
<th>(N \log N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n = 10)</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>(n = 100)</td>
<td>10,000</td>
<td>64</td>
</tr>
<tr>
<td>(n = 1,000)</td>
<td>100,000</td>
<td>9,965</td>
</tr>
<tr>
<td>(n = 10,000)</td>
<td>1,000,000</td>
<td>132,877</td>
</tr>
<tr>
<td>(n = 100,000)</td>
<td>10,000,000</td>
<td>1,465,964</td>
</tr>
<tr>
<td>(n = 1,000,000)</td>
<td>100,000,000</td>
<td>19,931,568</td>
</tr>
<tr>
<td>(n = 10,000,000)</td>
<td>1,000,000,000</td>
<td>232,534,966</td>
</tr>
<tr>
<td>(n = 100,000,000)</td>
<td>10,000,000,000</td>
<td>2,657,542,475</td>
</tr>
<tr>
<td>(n = 1,000,000,000)</td>
<td>100,000,000,000</td>
<td>29,897,352,854</td>
</tr>
</tbody>
</table>

WQU with Path Compression

On every call to `find`, compress all pointers on the path traversed so that they all point to the root.

\[ \text{find}(2) \]

Applications

- **Maze Generation**
  - From Algorithm Design and Applications, Goodrich & Tamassia

- **Percolation**
  - http://michaeltoth.net/blog/posts/2014/10/Percolation/
Percolation

Once we have a $N \times N$ grid, how can we check whether the system percolates?

1. first, build a union-find structure for the grid and 'union' open sites
2. second, check for 'connectivity' between sites in the top row with sites in the bottom row ($N^2$ queries)

Can we do better?