Imagine a World With No Good Algorithms

- Multimedia.  CD player, DVD, MP3, JPG, DivX, HDTV.
- Internet.  Packet routing, Google, Akamai.
- Information processing.  Database search, data compression.
- Computers.  Circuit layout, file system, compilers.
- Biology.  Human genome project, protein folding.
- Astrophysics.  N-body simulation.
- Transportation.  Airline crew scheduling, map routing.

Why Study Algorithms

- Using a computer?
  - Want it to go faster?  Process more data?
  - Want it to do something that would otherwise be impossible?

- Technology improves things by a constant factor.
  - But might be costly.
  - Good algorithmic design can do much better and might be cheap.
  - Supercomputers cannot rescue a bad algorithm.

- Algorithms as a field of study.
  - Old enough that basics are known.
  - New enough that new discoveries arise.
  - Burgeoning application areas.
  - Philosophical implications.
The Usual Suspects

Lectures: Kevin Wayne (Kevin)
- MW 11-12:20, CS 105.

Precepts: Nir Ailon (Nir), Miro Dudik (Miro)
- T 12:30, Friend 005.
- T 1:30, Friend 005.
- T 3:30, Friend 005.
- Clarify programming assignments, exercises, lecture material.
- First precept meets 2/10.

Coursework and Grading

Weekly programming assignments: 45%
- Due Thursdays 11:59pm, starting 2/12.

Weekly written exercises: 15%
- Due at beginning of Monday lecture, starting 2/9.

Exams:
- Closed book with cheatsheet.
- Midterm. 15%
- Final. 25%
- Staff discretion. Adjust borderline cases.

Course Materials

- Syllabus.
- Programming assignments.
- Exercises.
- Lecture notes.
- Old exams.

Algorithms in Java, 3rd edition.
- Parts 1-4 (COS 126 text).
- Part 5 (graph algorithms).

- Strings and geometry handouts.

Union Find

Quick find
Quick union
Weighted quick union
Path compression

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?

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 operations M can be huge.
- Number of objects N can be huge.

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>5</td>
<td>6</td>
<td>5–6</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>

Other Applications

More union-find applications.
- Hex.
- Percolation.
- Image processing.
- Minimum spanning tree.
- Least common ancestor.
- Equivalence of finite state automata.
- Compiling EQUIVALENCE statements in FORTRAN.
- Micali-Vazarani algorithm for nonbipartite matching.
- Weihe’s algorithm for edge-disjoint s-t paths in planar graphs.
- Scheduling unit-time tasks to P processors so that each job finishes between its release time and deadline.
- Scheduling unit-time tasks with a partial order to two processors in order to minimize last completion time.

References.
- A Linear Time Algorithm for a Special Case of Disjoint Set Union, Gabow and Tarjan.
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.
- Metallic sites in a composite system.
- When programming, convenient to name them 0 to N-1.
- When drawing, fun to use animals!

Quick-Find

Union(tiger, bear)
Quick-Find

Data structure.
- Maintain array \( \text{id}[] \) with name for each of \( N \) elements.
- If \( p \) and \( q \) are connected, then they have the same id.
- Initially, set id of each element to itself.

Find. To check if \( p \) and \( q \) are connected, see if they have same id.

\[
\text{for (int } i = 0; i < N; i++) \\
\text{ if (id}[i]\text{== pid) id}[i]\text{= id}[q];}
\]

Union. To merge components containing \( p \) and \( q \), change all entries with \( \text{id}[p] \) to \( \text{id}[q] \).

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

id[elephant] = skunk

Union(Elephant, Skunk)
Find with Quick-Union

Answer = Skunk

Find(Lobster)

Quick-Union

root(Tiger) = Elephant
root(Lobster) = Skunk
id[Elephant] = Skunk

Union(Tiger, Lobster)

Quick-Union

Data structure: disjoint forests.
- Maintain array \( id[] \) for each of \( N \) elements.
- Root of element \( x \) = \( id[id[...id[p]...]] \)

```
public int root(int x) {
    while (x != id[x])
        x = id[x];
    return x;
}
```

Find. Check if \( p \) and \( q \) have same root.

```
return (root(p) == root(q));
```

Union. Set the id of \( p \)'s root to \( q \)'s root.

```
int i = root(p);
int j = root(q);
id[i] = j;
```

keep going until it doesn't change

time proportional to depth of \( p \) and \( q \)

time proportional to depth of \( x \)
Weighted Quick-Union

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

Quick-union defect.
- Finding the root can be 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.

Data structure: disjoint forests.
- Also maintain array $sz[i]$ that counts the number of elements in the tree rooted at $i$.

Find. Same as quick union.

Union. Same as quick union, but merge smaller tree into the larger tree and update the $sz[]$ array.

Analysis.
- FIND takes time proportional to depth of $p$ and $q$ in tree.
- UNION takes constant time, given roots.
**Weighted Quick-Union**

Is performance improved?
- Theory: $\log N$ 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.
Path Compression

```
Find(Piggy)
```

Path Compression

```
Find(Piggy)
```

Weighted Quick-Union with Path Compression

Path compression.

- Add second loop to `root` to compress tree that sets the id of every examined node to the root.
- Simple one-pass variant: make each element point to grandparent.

```java
public int root(int x) {
    while (x != id[x]) {
        id[x] = id[id[x]]; // only one extra line of code
        x = id[x];
    }
    return x;
}
```

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

Another Application: Hex

Hex. (Piet Hein 1942, John Nash 1948, Parker Brothers 1962)
- Two players alternate in picking a cell in a hex grid.
- Black: make a black path from upper left to lower right.
- White: make a white path from lower left to upper right.
- Goal: algorithm to detect when a player has won?

Yet Another Application: Percolation

Percolation phase-transition.
- Two parallel conducting bars (top and bottom).
- Electricity flows from a site to one of its 4 neighbors if both are occupied by conductors.
- Suppose each site is randomly chosen to be a conductor or insulator with probability \( p \). What is percolation threshold \( p^* \) at which charge carriers can percolate from top to bottom?

~ 0.592746 for square lattices, but constant only known via simulation

Lessons

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick-find</td>
<td>( MN )</td>
</tr>
<tr>
<td>Quick-union</td>
<td>( MN )</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(M + N) )</td>
</tr>
</tbody>
</table>

Simple algorithms can be very useful.
- Start with brute force approach.
- Don’t use for large problems
- Can’t use for huge problems
- Strive for worst-case performance guarantees
- Identify fundamental abstractions: union-find, disjoint forests