What is COS 226?

- Intermediate-level survey course.
- Programming and problem solving with applications.
- Data structure: method to store information.

<table>
<thead>
<tr>
<th>topic</th>
<th>data structures and algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>data types</td>
<td>stack, queue, union-find, priority queue</td>
</tr>
<tr>
<td>sorting</td>
<td>quicksort, mergesort, heapsort, radix sorts</td>
</tr>
<tr>
<td>searching</td>
<td>hash table, BST, red-black tree</td>
</tr>
<tr>
<td>graphs</td>
<td>BFS, DFS, Prim, Kruskal, Dijkstra</td>
</tr>
<tr>
<td>strings</td>
<td>KMP, Regular expressions, TST, Huffman, LZW</td>
</tr>
<tr>
<td>geometry</td>
<td>Graham scan, k-d tree, Voronoi diagram</td>
</tr>
</tbody>
</table>

Why study algorithms?

Their impact is broad and far-reaching.

Internet.  Web search, packet routing, distributed file sharing, ...
Biology.  Human genome project, protein folding, ...
Computers.  Circuit layout, file system, compilers, ...
Computer graphics.  Movies, video games, virtual reality, ...
Security.  Cell phones, e-commerce, voting machines, ...
Multimedia.  CD player, DVD, MP3, JPG, DivX, HDTV, ...
Transportation.  Airline crew scheduling, map routing, ...
Physics.  N-body simulation, particle collision simulation, ...
...

Why study algorithms?

Old roots, new opportunities.

- Study of algorithms dates at least to Euclid
- Some important algorithms were discovered by undergraduates!
Why study algorithms?

To solve problems that could not otherwise be addressed.

Ex. Network connectivity. [stay tuned]

Why study algorithms?

For intellectual stimulation.

“For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing.” — Francis Sullivan

“An algorithm must be seen to be believed.” — D. E. Knuth

Why study algorithms?

They may unlock the secrets of life and of the universe.

Computational models are replacing mathematical models in scientific enquiry

\[
E = mc^2 \\
F = ma \\
F = \frac{Gm_1 m_2}{r^2} \\
- \frac{\hbar^2}{2m} \nabla^2 + V(r) \Psi(r) = E \Psi(r)
\]

20th century science (formula based)

21st century science (algorithm based)

“Algorithms: a common language for nature, human, and computer.” — Avi Wigderson

Why study algorithms?

For fun and profit.
Why study algorithms?

• Their impact is broad and far-reaching.
• Old roots, new opportunities.
• To solve problems that could not otherwise be addressed.
• For intellectual stimulation.
• They may unlock the secrets of life and of the universe.
• For fun and profit.

Why study anything else?

---

Course content.

8 programming assignments. 45%
• Available via course website.
• Due 11:55pm, starting Tuesday 2/12.
• Electronic submission.

Exams.
• Closed-book with cheatsheet.
• Midterm. 20%
• Final. 25%

Staff discretion. 10%
• Participation in lecture and precepts.
• Everyone needs to meet me at office hours!

---

The usual suspects

Lectures. Introduce new material, answer questions.

Precepts. Answer questions, solve problems, discuss programming assignment.

Coursework and grading

Resources (web)

Course content.
• Course info.
• Lecture slides.
• Programming assignments.

Course administration.
• Check grades.
• Submit assignments.

Booksites.
• Brief summary of content.
• Download code from lecture.

---

Resources (web)

http://www.princeton.edu/~cos226
http://www.cs.princeton.edu/IntroProgramming
http://www.cs.princeton.edu/algs4
Union-Find Algorithms

- dynamic connectivity
- quick find
- quick union
- improvements
- applications

Subtext of today's lecture (and this course)

Steps to developing a usable algorithm.
- Model the problem.
- Find an algorithm to solve it.
- Fast enough? Fits in memory?
- If not, figure out why.
- Find a way to address the problem.
- Iterate until satisfied.

The scientific method.

Mathematical analysis.

Questions

Not registered? You can't submit assignments until register in SCORE.

Change precept? Make the precept change in SCORE yourself; see Donna O'Leary (CS 410) for serious conflicts.
Dynamic connectivity

Given a set of objects
• Union: connect two objects.
• Find: is there a path connecting the two objects?

union(3, 4)
union(8, 0)
union(2, 3)
union(5, 6)

find(0, 2)  no
find(2, 4)  yes
union(5, 1)
union(7, 3)
union(1, 6)
union(4, 8)

find(0, 2)  yes
find(2, 4)  yes

Modeling the objects

Dynamic connectivity applications involve manipulating objects of all types.
• Variable name aliases.
• Pixels in a digital photo.
• Computers in a network.
• Web pages on the Internet.
• Transistors in a computer chip.
• Metallic sites in a composite system.

When programming, convenient to name objects 0 to N-1.
• Use integers as array index.
• Suppress details not relevant to union-find.
• Could use symbol table to translate from object names to integers.

Network connectivity: larger example

Network connectivity: larger example

Dynamic connectivity

Given a set of objects
• Union: connect two objects.
• Find: is there a path connecting the two objects?

union(3, 4)
union(8, 0)
union(2, 3)
union(5, 6)

find(0, 2)  no
find(2, 4)  yes
union(5, 1)
union(7, 3)
union(1, 6)
union(4, 8)

find(0, 2)  yes
find(2, 4)  yes
union(4, 8)

Modeling the objects

Dynamic connectivity applications involve manipulating objects of all types.
• Variable name aliases.
• Pixels in a digital photo.
• Computers in a network.
• Web pages on the Internet.
• Transistors in a computer chip.
• Metallic sites in a composite system.

When programming, convenient to name objects 0 to N-1.
• Use integers as array index.
• Suppress details not relevant to union-find.
• Could use symbol table to translate from object names to integers.
Transitivity. If \( p \) is connected to \( q \) and \( q \) is connected to \( r \), then \( p \) is connected to \( r \).

Connected components. Maximal subset of objects that are mutually connected.

Find query. Check if two objects are in the same subset.

Union command. Replace subsets containing two objects with their union.

Goal. Design efficient data structure for union-find.
- Find queries and union commands may be intermixed.
- Number of operations \( M \) can be huge.
- Number of objects \( N \) can be huge.

Public class UnionFind
- \( \text{UnionFind} \) (int \( N \)) create union-find data structure with \( N \) objects and no connections
- boolean \( \text{find} \) (int \( p \), int \( q \)) are \( p \) and \( q \) in the same subset?
- void \( \text{union} \) (int \( p \), int \( q \)) replace subsets containing \( p \) and \( q \) with their union

- Dynamic connectivity
- Quick find
- Quick union
- Improvements
- Applications
Data structure.
• Integer array \textit{id[]} of size \textit{N}.
• Interpretation: \textit{p} and \textit{q} are connected if they have the same id.

Quick-find [eager approach]

Check if \textit{p} and \textit{q} have the same id.

Union. To merge subsets containing \textit{p} and \textit{q}, change all entries with \textit{id}[\textit{p}] to \textit{id}[\textit{q}].

Quick-find example

\[
\begin{array}{cccccccccccc}
3-4 & 0 & 1 & 2 & 4 & 4 & 5 & 6 & 7 & 8 & 9 \\
4-9 & 0 & 1 & 2 & 9 & 9 & 5 & 6 & 7 & 8 & 9 \\
8-0 & 0 & 1 & 2 & 9 & 9 & 9 & 5 & 6 & 7 & 0 & 9 \\
2-3 & 0 & 1 & 9 & 9 & 9 & 5 & 6 & 7 & 0 & 9 \\
5-6 & 0 & 1 & 9 & 9 & 9 & 6 & 6 & 7 & 0 & 9 \\
5-9 & 0 & 1 & 9 & 9 & 9 & 9 & 9 & 7 & 0 & 9 \\
7-3 & 0 & 1 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 0 & 9 \\
4-8 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
6-1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

Quick-find: Java implementation

```java
public class QuickFind {
    private int[] id;
    public QuickFind(int N) {
        id = new int[N];
        for (int i = 0; i < N; i++)
            id[i] = i;
    }
    public boolean find(int p, int q) {
        return id[p] == id[q];
    }
    public void unite(int p, int q) {
        int pid = id[p];
        for (int i = 0; i < id.length; i++)
            if (id[i] == pid) id[i] = id[q];
    }
}
```
Quick-find is too slow

Quick-find defect.
- Union too expensive (N operations).
- Trees are flat, but too expensive to keep them flat.

Ex. May take $N^2$ operations to process $N$ union commands on $N$ objects.

<table>
<thead>
<tr>
<th>algorithm</th>
<th>union</th>
<th>find</th>
</tr>
</thead>
<tbody>
<tr>
<td>quick-find</td>
<td>N</td>
<td>1</td>
</tr>
</tbody>
</table>

Quadratic algorithms do not scale

Rough standard (for now).
- $10^9$ operations per second.
- $10^9$ words of main memory.
- Touch all words in approximately 1 second.

Ex. Huge problem for quick-find.
- $10^9$ union commands on $10^9$ objects.
- Quick-find takes more than $10^{18}$ operations.
- 30+ 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 [lazy approach]

Data structure.
- Integer array $id[i]$ of size $N$.
- Interpretation: $id[i]$ is parent of $i$.
- Root of $i$ is $id[id[...id[i]...]]$.

3's root is 9; 5's root is 6

Quick-find is too slow

Quick-find defect.
- Union too expensive (N operations).
- Trees are flat, but too expensive to keep them flat.

Ex. May take $N^2$ operations to process $N$ union commands on $N$ objects.

<table>
<thead>
<tr>
<th>algorithm</th>
<th>union</th>
<th>find</th>
</tr>
</thead>
<tbody>
<tr>
<td>quick-find</td>
<td>N</td>
<td>1</td>
</tr>
</tbody>
</table>

Quadratic algorithms do not scale

Rough standard (for now).
- $10^9$ operations per second.
- $10^9$ words of main memory.
- Touch all words in approximately 1 second.

Ex. Huge problem for quick-find.
- $10^9$ union commands on $10^9$ objects.
- Quick-find takes more than $10^{18}$ operations.
- 30+ 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 [lazy approach]

Data structure.
- Integer array $id[i]$ of size $N$.
- Interpretation: $id[i]$ is parent of $i$.
- Root of $i$ is $id[id[...id[i]...]]$.

3's root is 9; 5's root is 6
Quick-union [lazy approach]

Data structure.
- Integer array `id[]` of size `N`.
- Interpretation: `id[i]` is parent of `i`.
- Root of `i` is `id[id[...id[i]...]]`.

Find. Check if `p` and `q` have the same root.

Union. To merge subsets containing `p` and `q`, set the id of `q`'s root to the id of `p`'s root.

Quick-union: Java implementation

```java
public class QuickUnion {
    private int[] id;
    public QuickUnion(int N) {
        id = new int[N];
        for (int i = 0; i < N; i++) id[i] = i;
    }
    private int root(int i) {
        while (i != id[i]) i = id[i];
        return i;
    }
    public boolean find(int p, int q) {
        return root(p) == root(q);
    }
    public void unite(int p, int q) {
        int i = root(p), j = root(q);
        id[i] = j;
    }
}
```

Quick-find defect.
- Union too expensive (`N` operations).
- Trees are flat, but too expensive to keep them flat.

Quick-union defect.
- Trees can get tall.
- Find too expensive (could be `N` operations).

Quick-find is also too slow

<table>
<thead>
<tr>
<th>algorithm</th>
<th>union</th>
<th>find</th>
</tr>
</thead>
<tbody>
<tr>
<td>quick-find</td>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>quick-union</td>
<td>N*</td>
<td>N</td>
</tr>
</tbody>
</table>

* includes cost of finding root
Weighted quick-union: Java implementation

Data structure. Same as quick-union, but maintain extra array \( \text{sz}[i] \) to count number of objects in the tree rooted at \( i \).

Find. Identical to quick-union.

\[
\text{return root}(p) == \text{root}(q);
\]

Union. Modify quick-union to:
- Merge smaller tree into larger tree.
- Update the \( \text{sz}[i] \) array.

```java
int i = root(p), j = root(q);
if (sz[i] < sz[j]) { id[i] = j; sz[j] += sz[i]; }
else { id[j] = i; sz[i] += sz[j]; }
```
Weighted quick-union analysis

Analysis.
- Find: takes time proportional to depth of p and q.
- Union: takes constant time, given roots.
- Fact: depth is at most \( \log N \). [needs proof]

Q. Stop at guaranteed acceptable performance?
A. No, easy to improve further.

Quick union with path compression.
Just after computing the root of \( p \), set the id of each examined node to \( \text{root}(p) \).

Path compression: Java implementation

Standard implementation: add second loop to \( \text{root}() \) to set the id of each examined node to the root.

Simpler one-pass variant: make every other node in path point to its grandparent.

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

In practice. No reason not to! Keeps tree almost completely flat.

Weighted quick-union with path compression example
Theorem. [Tarjan 1975] Starting from an empty data structure, any sequence of \(M\) union and find operations on \(N\) objects takes \(O(N + M \lg^* N)\) time.
- Proof is very difficult.
- But the algorithm is still simple!

Linear algorithm?
- Cost within constant factor of reading in the data.
- In theory, WQUPC is not quite linear.
- In practice, WQUPC is linear.

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

Amazing fact. No linear-time linking strategy exists.

WQUPC performance

<table>
<thead>
<tr>
<th>(N)</th>
<th>(\lg^* N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<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>

WQUPC makes it possible to solve problems that could not otherwise be addressed.

Ex. [10^9 unions and finds with 10^9 objects]
- WQUPC reduces time from 30 years to 6 seconds.
- Supercomputer won’t help much; good algorithm enables solution.

Summary

Bottom line. WQUPC makes it possible to solve problems that could not otherwise be addressed.

<table>
<thead>
<tr>
<th>algorithm</th>
<th>worst-case time</th>
</tr>
</thead>
<tbody>
<tr>
<td>quick-find</td>
<td>(M N)</td>
</tr>
<tr>
<td>quick-union</td>
<td>(M N)</td>
</tr>
<tr>
<td>weighted QU</td>
<td>(N + M \log N)</td>
</tr>
<tr>
<td>QU + path compression</td>
<td>(N + M \log N)</td>
</tr>
<tr>
<td>weighted QU + path compression</td>
<td>((M + N) \lg^* N)</td>
</tr>
</tbody>
</table>

M union-find operations on a set of \(N\) objects

Union-find applications

- Percolation.
- Games (Go, Hex).
- Image processing.
✓ Network connectivity.
- Least common ancestor.
- Equivalence of finite state automata.
- Hinley-Milner polymorphic type inference.
- Kruskal’s minimum spanning tree algorithm.
- Compiling equivalence statements in Fortran.
- Morphological attribute openings and closings.
A model for many physical systems:
• N-by-N grid of sites.
• Each site is open with probability $p$ (or blocked with probability $1-p$).
• System **percolates** if top and bottom are connected by open sites.

<table>
<thead>
<tr>
<th>model</th>
<th>system</th>
<th>vacant site</th>
<th>occupied site</th>
<th>percolates</th>
</tr>
</thead>
<tbody>
<tr>
<td>electricity</td>
<td>material</td>
<td>conductor</td>
<td>insulated</td>
<td>conducts</td>
</tr>
<tr>
<td>fluid flow</td>
<td>material</td>
<td>empty</td>
<td>blocked</td>
<td>porous</td>
</tr>
<tr>
<td>social interaction</td>
<td>population</td>
<td>person</td>
<td>empty</td>
<td>communicates</td>
</tr>
</tbody>
</table>

Depends on site vacancy probability $p$.

**Likelihood of percolation**
Theory guarantees a sharp threshold \( p^* \).

- \( p > p^* \): almost certainly percolates.
- \( p < p^* \): almost certainly does not percolate.

Q. What is the value of \( p^* \)?

---

**Percolation phase transition**

---

**Monte Carlo simulation**

- Initialize whole grid to be blocked.
- Make random sites open until top connected to bottom.
- Vacancy percentage estimates \( p^* \).

---

**UF solution to find percolation threshold**

How to check whether system percolates?

- Create object for each site (and virtual top and bottom objects).
- Sites are in same set if connected by vacant sites.
- Percolates if top and bottom row are in same set.

---

**UF solution to find percolation threshold**

Q. How to make declare a new site vacant?
Q. How to make declare a new site vacant?
A. Take union of site and all adjacent vacant sites.

Percolation threshold

Q. What is percolation threshold $p^*$?
A. About 0.592746 for large square lattices.

Steps to developing a usable algorithm.
• Model the problem.
• Find an algorithm to solve it.
• Fast enough? Fits in memory?
• If not, figure out why.
• Find a way to address the problem.
• Iterate until satisfied.

Subtext of today's lecture (and this course)

The scientific method.

Mathematical analysis.