CSE 421: Introduction to Algorithms

I: Overview

Fall 2011
Anna Karlin

Administrivia

People:
- Anna Karlin
- Johnny Yan

All relevant course information:
http://www.cs.washington.edu/421
- Office hours, Wednesday 4-5, CSE 216

We will cover a good part of chapters 1-8.
Slides by combination of Larry Ruzzo, Kevin Wayne and others.

Administrivia

- Weekly homework, due Thursday ~ 40%
- Take home midterm, out Nov 10, due Nov 17 ~25%
- In-class, open book, open notes final ~35%

Working on homework sets:
- Collaboration on formulation of ideas allowed.
- Writing up solutions – can submit jointly with one other person.
- You may not consult written materials other than the course materials.
- We prefer that homework solutions be typed.
- Please indicate on your homework all people that you discussed the problems with, and indicate any and all sources you used.
- See grading guidelines handout.

What the course is about

Design of Algorithms
- design methods
- common or important types of problems
- analysis of algorithms - efficiency
- correctness proofs
What the course is about

Complexity, NP-completeness and intractability
solving problems in principle is not enough
algorithms must be efficient
some problems have no efficient solution
NP-complete problems
important & useful class of problems whose solutions
(seemingly) cannot be found efficiently, but can be
checked easily

Very Rough Division of Time

Algorithms (7 weeks)
Analysis of Algorithms
Basic Algorithmic Design Techniques
Graph Algorithms
Complexity & NP-completeness (2 weeks)

Check online
calendar page for
(evolving) details

Complexity Example

Cryptography (e.g. RSA, SSL in browsers)
Secret: p,q prime, say 512 bits each
Public: n which equals p x q, 1024 bits
In principle
there is an algorithm that given n will find p and q:
try all $2^{512} > 1.3 \times 10^{154}$ possible p's: kinda slow…
In practice
no fast algorithm known for this problem (on non-quantum computers)
security of RSA depends on this fact
("quantum computing": strongly driven by possibility of changing this)

Algorithms versus Machines

We all know about Moore’s Law and the
exponential improvements in hardware...

Ex: sparse linear equations over 25 years

10 orders of magnitude improvement!
Algorithms or Hardware?

25 years progress solving sparse linear systems

hardware: 4 orders of magnitude

Source: Sandia, via M. Schultz

Algorithms or Hardware?

25 years progress solving sparse linear systems

hardware: 4 orders of magnitude

software: 6 orders of magnitude

Source: Sandia, via M. Schultz

Algorithms or Hardware?

The N-Body Problem:

in 30 years $10^7$ hardware $10^{10}$ software

Source: T. Quinn

Algorithm: definition

Procedure to accomplish a task or solve a well-specified problem

Well-specified: know what all possible inputs look like and what output looks like given them “accomplish” via simple, well-defined steps

Ex: sorting names (via comparison)

Ex: checking for primality (via $+, -, *, /, \leq$)
Algorithms: a sample problem

Printed circuit-board company has a robot arm that solders components to the board.
Time: proportional to total distance the arm must move from initial rest position around the board and back to the initial position.
For each board design, find best order to do the soldering.

Printed Circuit Board

A Well-defined Problem

Input: Given a set $S$ of $n$ points in the plane.
Output: The shortest cycle tour that visits each point in the set $S$.

Better known as “TSP”

How might you solve it?
Nearest Neighbor Heuristic

Start at some point $p_0$
Walk first to its nearest neighbor $p_1$
Repeatedly walk to the nearest unvisited neighbor $p_2$, then $p_3$, ... until all points have been visited
Then walk back to $p_0$

**Nearest Neighbor Heuristic**

An input where it works badly

$$\text{length } \sim 84$$

An input where it works badly

$$\text{optimal soln for this example }\text{ length } = 63.8$$
Revised idea - Closest pairs first

Repeatedly join the closest pair of points (s.t. result can still be part of a single loop in the end. I.e., join endpoints, but not points in middle, of path segments already created.)

How does this work on our bad example?

Another bad example

Another bad example

Something that works

"Brute Force Search": For each of the $n! = n(n-1)(n-2)...1$ orderings of the points, check the length of the cycle you get. Keep the best one.
Two Notes

The two incorrect algorithms were greedy

- Often very natural & tempting ideas
- They make choices that look great “locally” (and never reconsider them)
- When greed works, the algorithms are typically efficient
- BUT: often does not work - you get boxed in

Our correct alg avoids this, but is incredibly slow

- $20!$ is so large that checking one billion orderings per second would take 2.4 billion seconds (around 70 years!)
- And growing: $n! \sim \sqrt{2 \pi n} \cdot (n/e)^n \sim 2^{O(n \log n)}$

Something that “works” (differently)

1. Find Min Spanning Tree

Something that “works” (differently)

2. Walk around it

Something that “works” (differently)

3. Take shortcuts (instead of revisiting)
Something that “works” (differently): Guaranteed Approximation

Does it seem wacky?
Maybe, but it's always within a factor of 2 of the best tour!
- deleting one edge from best tour gives a spanning tree, so $\text{Min spanning tree} < \text{best tour}
- \text{best tour} \leq \text{wacky tour} \leq 2 * \text{MST} < 2 * \text{best}

\[\text{triangle inequality}\]

The Morals of the Story

- Algorithms are important
  - Many performance gains outstrip Moore’s law
- Simple problems can be hard
  - Factoring, TSP
- Simple ideas don’t always work
  - Nearest neighbor, closest pair heuristics
- Simple algorithms can be very slow
  - Brute-force factoring, TSP
- Changing your objective can be good
  - Guaranteed approximation for TSP
- And: for some problems, even the best algorithms are slow