CSE 421 Course Introduction

- CSE 421, Introduction to Algorithms
  - MWF, 1:30-2:20 pm
  - MGH 421

- Instructor
  - Richard Anderson, anderson@cs.washington.edu
  - Office hours:
    - CSE 582
    - Office hours TBD

- Teaching Assistants
  - Cyrus Rashtchian
  - Yueqi Sheng
  - Erin Yoon
  - Kuai Yu
Announcements

• It’s on the web.
• Homework due Wednesdays
  – HW 1, Due October 7, 2015
  – It’s on the web (or will be soon)
• You should be on the course mailing list
  – But it will probably go to your uw.edu account
Text book

• Algorithm Design
• Jon Kleinberg, Eva Tardos

• Read Chapters 1 & 2

• Expected coverage:
  – Chapter 1 through 7
Course Mechanics

• Homework
  – Due Wednesdays
  – About 5 problems, sometimes programming
  – Target: 1 week turnaround on grading

• Exams (In class)
  – Midterm, Monday, November 2 (probably)
  – Final, Monday, December 14, 2:30-4:20 pm

• Approximate grade weighting
  – HW: 50, MT: 15, Final: 35

• Course web
  – Slides, Handouts
All of Computer Science is the Study of Algorithms
How to study algorithms

• Zoology
• Mine is faster than yours is
• Algorithmic ideas
  – Where algorithms apply
  – What makes an algorithm work
  – Algorithmic thinking
Introductory Problem: Stable Matching

• Setting:
  – Assign TAs to Instructors
  – Avoid having TAs and Instructors wanting changes
    • E.g., Prof A. would rather have student X than her current TA, and student X would rather work for Prof A. than his current instructor.
Formal notions

- Perfect matching
- Ranked preference lists
- Stability
Example (1 of 3)

$m_1: w_1 \ w_2$

$m_2: w_2 \ w_1$

$w_1: m_1 \ m_2$

$w_2: m_2 \ m_1$
Example (2 of 3)

\[ m_1: w_1 \; w_2 \]
\[ m_2: w_1 \; w_2 \]
\[ w_1: m_1 \; m_2 \]
\[ w_2: m_1 \; m_2 \]
Example (3 of 3)

\[ m_1: w_1 \ w_2 \]
\[ m_2: w_2 \ w_1 \]
\[ w_1: m_2 \ m_1 \]
\[ w_2: m_1 \ m_2 \]
Formal Problem

• Input
  – Preference lists for \( m_1, m_2, \ldots, m_n \)
  – Preference lists for \( w_1, w_2, \ldots, w_n \)

• Output
  – Perfect matching \( M \) satisfying stability property:

\[
\text{If } (m', w') \in M \text{ and } (m'', w'') \in M \text{ then } (m' \text{ prefers } w' \text{ to } w'') \text{ or } (w'' \text{ prefers } m'' \text{ to } m')
\]
Idea for an Algorithm

m proposes to w

If w is unmatched, w accepts
If w is matched to m₂
  If w prefers m to m₂ w accepts m, dumping m₂
  If w prefers m₂ to m, w rejects m

Unmatched m proposes to the highest w on its preference list that it has not already proposed to
Algorithm

Initially all \( m \) in \( M \) and \( w \) in \( W \) are free
While there is a free \( m \)
    \( w \) highest on \( m \)'s list that \( m \) has not proposed to
    if \( w \) is free, then match \((m, w)\)
    else
        suppose \((m_2, w)\) is matched
        if \( w \) prefers \( m \) to \( m_2 \)
            unmatch \((m_2, w)\)
            match \((m, w)\)
Example

\[ m_1 : w_1 \ w_2 \ w_3 \]
\[ m_2 : w_1 \ w_3 \ w_2 \]
\[ m_3 : w_1 \ w_2 \ w_3 \]
\[ w_1 : m_2 \ m_3 \ m_1 \]
\[ w_2 : m_3 \ m_1 \ m_2 \]
\[ w_3 : m_3 \ m_1 \ m_2 \]
Does this work?

• Does it terminate?
• Is the result a stable matching?

• Begin by identifying invariants and measures of progress
  – m’s proposals get worse (have higher m-rank)
  – Once w is matched, w stays matched
  – w’s partners get better (have lower w-rank)
Claim: If an m reaches the end of its list, then all the w’s are matched.
Claim: The algorithm stops in at most $n^2$ steps
When the algorithms halts, every $w$ is matched

Why?

Hence, the algorithm finds a perfect matching
The resulting matching is stable

Suppose

\[(m_1, w_1) \in M, (m_2, w_2) \in M\]

\[m_1 \text{ prefers } w_2 \text{ to } w_1\]

How could this happen?
Result

• Simple, $O(n^2)$ algorithm to compute a stable matching

• Corollary
  – A stable matching always exists
A closer look

Stable matchings are not necessarily fair

\[ \begin{align*}
    m_1 &: w_1 \ w_2 \ w_3 \\
    m_2 &: w_2 \ w_3 \ w_1 \\
    m_3 &: w_3 \ w_1 \ w_2 \\
    w_1 &: m_2 \ m_3 \ m_1 \\
    w_2 &: m_3 \ m_1 \ m_2 \\
    w_3 &: m_1 \ m_2 \ m_3
\end{align*} \]

How many stable matchings can you find?
Algorithm under specified

• Many different ways of picking m’s to propose
• Surprising result
  – All orderings of picking free m’s give the same result

• Proving this type of result
  – Reordering argument
  – Prove algorithm is computing something mores specific
    • Show property of the solution – so it computes a specific stable matching
Proposal Algorithm finds the best possible solution for M

Formalize the notion of best possible solution:

(m, w) is valid if (m, w) is in some stable matching

best(m): the highest ranked w for m such that (m, w) is valid

S* = {(m, best(m)}

Every execution of the proposal algorithm computes S*
Proof

See the text book – pages 9 – 12

Related result: Proposal algorithm is the worst case for W
Algorithm is the M-optimal algorithm
Proposal algorithms where w’s propose is W-Optimal
Best choices for one side may be bad for the other

Design a configuration for problem of size 4:

M proposal algorithm:
All m’s get first choice, all w’s get last choice

W proposal algorithm:
All w’s get first choice, all m’s get last choice

m₁:

m₂:

m₃:

m₄:

w₁:

w₂:

w₃:

w₄:
But there is a stable second choice

Design a configuration for problem of size 4:

M proposal algorithm:
All m’s get first choice, all w’s get last choice

W proposal algorithm:
All w’s get first choice, all m’s get last choice

There is a stable matching where everyone gets their second choice
Key ideas

• Formalizing real world problem
  – Model: graph and preference lists
  – Mechanism: stability condition

• Specification of algorithm with a natural operation
  – Proposal

• Establishing termination of process through invariants and progress measure

• Under specification of algorithm

• Establishing uniqueness of solution