CSE 421
Algorithms

Richard Anderson

Winter 2019

Lecture 1

CSE 421 Course Introduction

- CSE 421, Introduction to Algorithms
  - MWF, 1:30-2:20 pm
  - THO 101
- Instructor
  - Richard Anderson, anderson@cs.washington.edu
  - Office hours:
    - CSE 582 (Until Feb 18, then CSE2 344)
    - Office hours: Monday 2:30-3:30, Wednesday 2:30-3:30
  - Getting back from India tomorrow
- Teaching Assistants
  - Sean Jaffe
  - Mathew Luo
  - Aditya Saraf
  - Xin Yang
  - Faye Yu
  - Leiyi Zhang

Announcements

- It's on the web.
- Homework due Wednesdays
  - HW 1, Due January 16
  - It's on the web
  - Submit solutions on canvas
- You should be on the course mailing list
  - But it will probably go to your uw.edu account

Textbook

- Algorithm Design
- Jon Kleinberg, Eva Tardos
- Read Chapters 1 & 2
- Expected coverage:
  - Chapter 1 through 7
- Book available at:
  - UW Bookstore ($174)
  - Ebay ($17.95 to $264.54)
  - Amazon ($17.83 and up)
  - Kindle ($105.99)
  - PDF (Google Search)

There is only one edition of this book, and the international versions are fine

Course Mechanics

- Homework
  - Due Wednesdays
  - About 5 problems, sometimes programming
  - Target: 1 week turnaround on grading
- Exams (In class)
  - Midterm, Wednesday, February 13
  - Final, Monday, March 18, 2:30-4:20 pm
- Approximate grade weighting
  - HW: 50, MT: 15, Final: 35
- Course web
  - Slides, Handouts

Guest Lecturer

- Robbie Weber

Introductory Problem

For today: A sample of what you'll do in this course
- Define a problem
- And solve it with an algorithm

Motivation:
- Assign TAs to Instructors
- Avoid a TA and Instructor wanting the same change
  - 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.

Motivation

The real world is complicated.
- Students shouldn't TA a course they haven't taken.
- Instructors need varying numbers of TAs.
- There are more TA applicants than positions.

Let's simplify
- Everyone can be assigned to every course.
- Every course needs exactly one TA.
- There are exactly as many applicants as courses.

The algorithmic ideas in the simple case can be adapted to handle the general case.

Stable Matching Problem

In the simplified version, a metaphor other than TAs and courses is common.

We'll think of the sets as "men" and "women" instead of courses and TAs.

The problem design is really used to
- Match new doctors to their residencies
- Assign students to high schools in some districts
- Among many other applications.

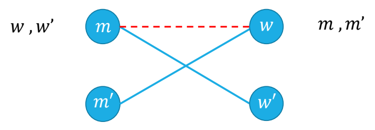
Stable Matching Problem

Given two sets $M = \{m_1, \dots, m_n\}, W = \{w_1, \dots, w_n\}$ each ranks **every** person in the other set.

Goal: Match each person to **exactly one** person in the other set, respecting their preferences.

How do we "respect preferences"?

Avoid unmatched pairs (m, w) where m prefers w to his match, and w prefers m to her match.



Stable Matching, More Formally

Perfect matching:
- Each man is paired with exactly one woman.
- Each woman is paired with exactly one man.

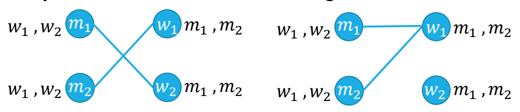
Stability: no incentive to exchange
- an unmatched pair m-w is unstable
- if man m and woman w prefer each other to current partners.

Stable matching: perfect matching with no unstable pairs.

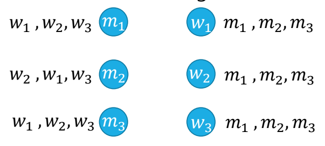
Stable Matching Problem
Given: the preference lists of n men and n women
Find: a stable matching (if one exists).

Examples

Why are these not stable matchings?



Find a stable matching for this instance.



Questions

Does a stable matching always exist?

Can we find a stable matching efficiently?

We'll answer both of those questions today

Let's start with the second one.

Idea for an Algorithm

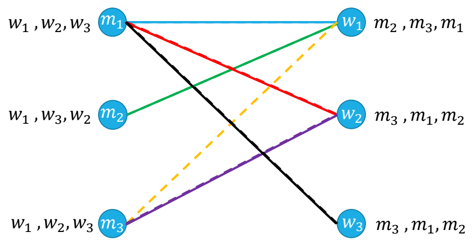
```
$m$ proposes to $w$
 If $w$ is unmatched
   $w$ tentatively accepts
 Else $w$ is matched to some man $m'$
   If $w$ prefers $m$ to $m'$
     $w$ tentatively accepts $m$, rejecting $m'$
   Else ($w$ prefers $m'$ to $m$)
     $w$ rejects $m$
```

Unmatched m proposes to the highest w on his preference list that he 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 $(m', w)$ are matched
        if $w$ prefers $m$ to $m'$
            unmatch $(m', w)$
            match $(m, w)$
```

Algorithm Example



Proposals: $m_1, m_2, m_1, m_3, m_3, m_1$

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
- Once w is matched, w stays matched
- w's partners get better

Claim 1: If m reaches the end of his list, then all the women are matched

A woman only rejects a proposal when she gets a better one, so if she rejects someone, she must have a match from then on.

So everyone on m's list must be matched.

And every woman is on m's list!

Claim 2: The algorithm stops in $O(n^2)$ steps

If every woman is matched, every man must be matched too.

- Because each woman is matched to exactly one man and there are an equal number of men and women.

It takes at most $O(n^2)$ proposals to get to the end of some man's list.

Claim 2 now follows from Claim 1.

Question 1 answered: The algorithm halts (quickly)!

Now question 2: does it produce a stable matching?

Claim 3: When the algorithms halts, every woman is matched

Why?

If we reach the end of someone's list, it follows from Claim 1.

If we don't, then every man is matched, but every man is matched to exactly one woman.

Hence, the algorithm finds a perfect matching

Claim 4: The resulting matching is stable.

We want to prove a negative

there is no unstable pair.

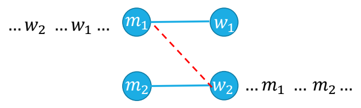
That's a good sign for proof by contradiction.

Suppose (for contradiction) that (m_1, w_1) and (m_2, w_2) are matched, but

m_1 prefers w_2 to w_1 and

w_2 prefers m_1 to m_2

Claim 4: The resulting matching is stable.



How did m_1 end up matched to w_1?

He must have proposed to and been rejected by w_2.

Why did w_2 reject m_1? She got a better offer from m'.

If w_2 ever changed partners after that, it only got better for her, so she must prefer m_2 (her final match) to m_1.

A contradiction!

Result

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

Corollary

- A stable matching always exists

The corollary isn't obvious!

The "stable roommates problem" doesn't always have a solution:

- $2n$ people, rank the other $2n - 1$
- Goal is to pair them without any unstable pairs.

Next Time

The algorithm is a bit underspecified.

- when more than one man is unmatched, does it matter who goes first?

A stable matching always exists, but can there be more than one?

- For example, what happens if the women propose instead of the men?