CSE 421
Introduction to Algorithms

Lecture 1: Intro & Stable Matching

https://cs.washington.edu/421

Instructor

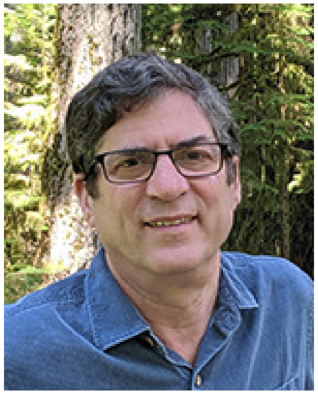
Paul Beame [he/him]

beame@cs

Specialty: **Complexity and Applications**

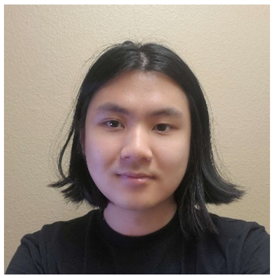
https://homes.cs.washington.edu/~beame

Office: CSE 668



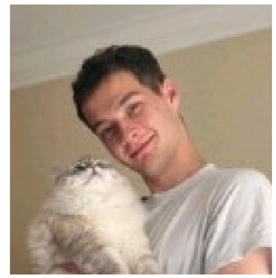
A Dedicated Team of TAs

Daniel Gao

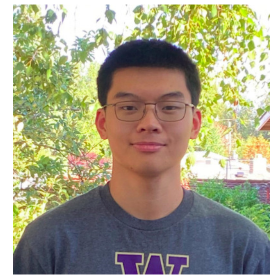


Raymond Guo

Samarjit Kaushik

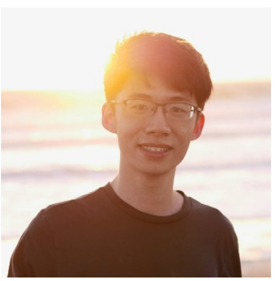


Kyle Mumma



Edward Qin

Robert Stevens



Glenn Sun

Aman Thukral

Tom Tian

Maxwell Wang

Ben Zhang

Muru Zhang

See https://cs.washington.edu/421/staff.html to learn more about their backgrounds and interests!

Getting Started (Your TODO List)

- Make sure you are on Ed (a.k.a. EdStem)!
  - Check your inbox – and maybe your SPAM filter – for an invitation
- Attend your first Quiz Section tomorrow!
- Homework 1 will be out tonight
  - You will have enough to start on it after section tomorrow
  - Start thinking about it right away after that
- Get all the credit you deserve: Sign up for CSE 493Z

  Optional
- Attend lecture and participate
  - Students who participate do better on average

Coursework

- 8 homework assignments roughly (due Wednesdays)
  - Typically 1 mechanical problem
    3 long-form problems
  - See the Homework Guide linked on the course website
  - Start early to reduce amount of time you need to concentrate on them
    - Use your brain's background processing
  - OK to talk with fellow students but solution write-up must be your own
    - See syllabus https://cs.washington.edu/421/syllabus.html
  - Use of outside resources for solutions **forbidden** (see syllabus)
    - Generative AI does worse than almost anyone in the class would on their own…

Late Problem Days

- Late days per problem rather than for the whole assignment
  - Each problem is a separate Gradescope submission
  - Max 2 late days per problem; limit on total # of late problem days
  - You should submit anything that is done as soon as you are finished with it
  - See the syllabus for details

Exam dates

Midterm: Wednesday Nov 8 (possibly evening to give you more time for the same problems)

Final Exam: Standard exam time and place: Monday Dec 11, 2:30-4:20 here

Grading scheme

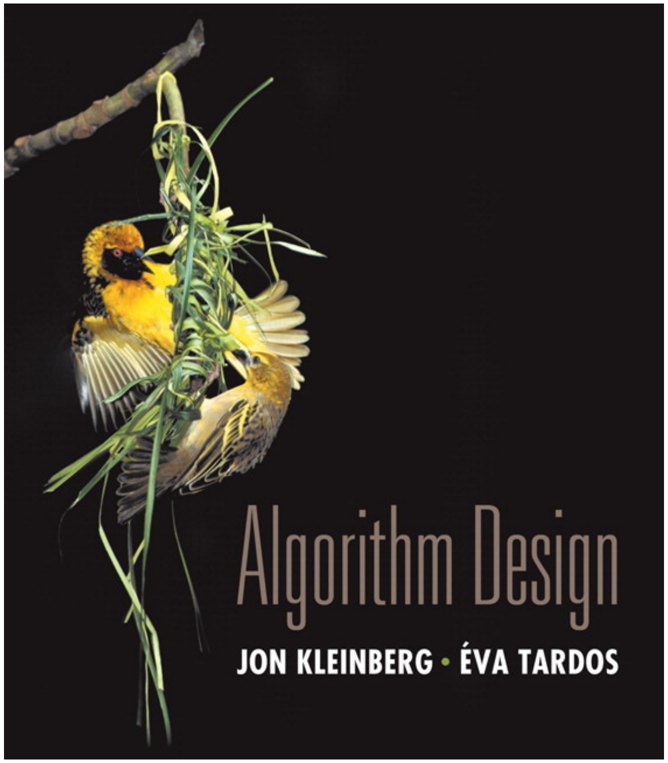
- Homework 55%
- Midterm 15-20%
- Final Exam 25-30%

Textbook

Kleinberg-Tardos: Algorithm Design

- International Edition just as good
- Plus supplements on website

- Worth reading
  - Good for reading sequentially and learning how to think like an algorithm designer
  - Not as good for random access
- Not required
  - All required content will be on slides in lectures and quiz section



Introduction to Algorithms

- Basic techniques for the design and analysis of algorithms.
  - Develop a toolkit of ways to find efficient algorithms to solve problems
  - Prove that the algorithms are correct
  - Analyze their efficiency properties
  - Communicate these algorithms and their properties to others

On efficiency

- Originally, efficiency was important for many reasons but partly because computers were weak

- Now we have powerful computers but
  - Data has grown to be enormous
    - We need *even more* efficient algorithms at this scale
  - Computation has an *energy cost* and represents a significant part of society's total energy use
    - Efficient computing is essential to reducing that cost
  - Additional power is of little help for inefficient (e.g. brute force) solutions

Introduction to Algorithms

- Stable Matching

Matching Medical Residents to Hospitals

Goal: Given a set of preferences among hospitals and medical school residents (graduating medical students), design a *self-reinforcing* admissions process.

not matched

Unstable pair: applicant x and hospital y are *unstable* if:

- x prefers y to their assigned hospital.
- y prefers x to one of its admitted residents.

Stable assignment. Assignment with no unstable pairs.

- Natural and desirable condition.
- Individual self-interest will prevent any applicant/hospital side deal from being made.

Simpler: Stable Matching Problem

Goal: Given two groups of n people each, find a "suitable" matching.

- Participants rate members from opposite group.
- Each person lists members from the other group in order of preference from best to worst.

	1st (favorite)	2nd	3rd (least favorite)
X	A	B	C
Y	B	A	C
Z	A	B	C

*Group **P** Preference Profile*

	1st (favorite)	2nd	3rd (least favorite)
A	Y	X	Z
B	X	Y	Z
C	X	Y	Z

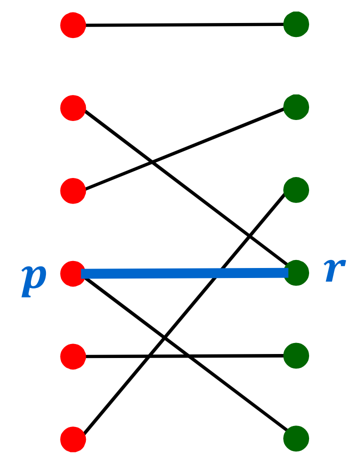
*Group **R** Preference Profile*

Stable Matching Problem

Perfect matching: everyone is matched to precisely one person from the other group

Stability: self-reinforcing, i.e. no incentive to undermine assignment by joint action.

- For a matching M, an unmatched pair p-r from different groups is *unstable* if p and r prefer each other to current partners.
- Unstable pair p-r could each improve by ignoring the assignment.



Stable matching: perfect matching with no unstable pairs.

Stable matching problem: Given the preference lists of n people from each of two groups, find a stable matching between the two groups if one exists.

Stable Matching Problem

Q: Is matching (X,C), (Y,B), (Z,A) stable?

No

unstable

X would prefer A
A would prefer X

not the only case

	favorite 1st	2nd	least favorite 3rd
X	A	B	C
Y	B	A	C
Z	A	B	C

*Group **P** Preference Profile*

	favorite 1st	2nd	least favorite 3rd
A	Y	X	Z
B	X	Y	Z
C	X	Y	Z

*Group **R** Preference Profile*

Stable Matching Problem

Q: Is matching (X,C), (Y,B), (Z,A) stable?

A: No. B and X prefer each other.

	1st (favorite)	2nd	3rd (least favorite)
X	A	B	C
Y	B	A	C
Z	A	B	C

*Group **P** Preference Profile*

	1st (favorite)	2nd	3rd (least favorite)
A	Y	X	Z
B	X	Y	Z
C	X	Y	Z

*Group **R** Preference Profile*

Stable Matching Problem

Q: Is matching (X,A), (Y,B), (Z,C) stable?

Yes X and Y got 1st choice so want no in unstable pair
Nobody in group R prefers Z.

	1st (favorite)	2nd	3rd (least favorite)
X	A	B	C
Y	B	A	C
Z	A	B	C

*Group **P** Preference Profile*

	1st (favorite)	2nd	3rd (least favorite)
A	Y	X	Z
B	X	Y	Z
C	X	Y	Z

*Group **R** Preference Profile*

Stable Matching Problem

Q: Is matching (X,A), (Y,B), (Z,C) stable?

A: Yes

favorite ↓ … least favorite ↓

	1st	2nd	3rd
X	A	B	C
Y	B	A	C
Z	A	B	C

*Group **P** Preference Profile*

favorite ↓ … least favorite ↓

	1st	2nd	3rd
A	Y	X	Z
B	X	Y	Z
C	X	Y	Z

*Group **R** Preference Profile*

Variant: Stable Roommate Problem

Q. Do stable matchings always exist?

A. Not obvious a priori.

Stable roommate problem:

- $2n$ people; each person ranks others from 1 to $2n-1$.
- Assign roommate pairs so that no unstable pairs.

	1st	2nd	3rd
A	B	C	D
B	C	A	D
C	A	B	D
D	A	B	C

(A,B), (C,D) $\Rightarrow$ B-C unstable
(A,C), (B,D) $\Rightarrow$ A-B unstable
(A,D), (B,C) $\Rightarrow$ A-C unstable

Observation: Stable matchings do not always exist for stable roommate problem.

Propose-And-Reject Algorithm

Propose-and-reject algorithm: [Gale-Shapley 1962]

Intuitive method that guarantees to find a stable matching.

- Members of one group ***P*** make *proposals*, the other group ***R*** *receives* proposals

```
Initialize each person to be free
while (some p in P is free) {
    Choose some free p in P
    r = 1st person on p's preference list to whom p has not yet proposed
    if (r is free)
        tentatively match (p,r)    //p and r both engaged, no longer free
    else if (r prefers p to current tentative match p')
        replace (p',r) by (p,r)    //p now engaged, p' now free
    else
        r rejects p
}
```

Proof of Correctness: Termination (not obvious from the code)

Observation 1: Members of P propose in decreasing order of preference.

Claim: The Gale-Shapley Algorithm terminates after at most n^2 iterations.

Proof: Proposals are never repeated (by Observation 1) and there are only n^2 possible proposals. ■

It could be nearly that bad…

General form of this example will take $n(n-1)+1$ proposals.

	1st	2nd	3rd	4th	5th
V	A	B	C	D	E
W	B	C	D	A	E
X	C	D	A	B	E
Y	D	A	B	C	E
Z	A	B	C	D	E

Preference Profile for P

	1st	2nd	3rd	4th	5th
A	W	X	Y	Z	V
B	X	Y	Z	V	W
C	Y	Z	V	W	X
D	Z	V	W	X	Y
E	V	W	X	Y	Z

Preference Profile for R

Proof of Correctness: Perfection

Observation 2: Once a member of R is matched, they never become free; they only "trade up.“

Claim: Everyone gets matched.

Proof:

- After some p proposes to the last person on their list, all the r in R have been proposed to by someone (by p at least).
- By Observation 2, every r in R is matched at that point.
- Since $|P| = |R|$ every p in P is also matched. ■

Proof of Correctness: Stability

Claim: No unstable pairs in the final Gale-Shapley matching $\boldsymbol{M}$

Proof: Consider a pair $\boldsymbol{p}$-$\boldsymbol{r}$ not matched by $\boldsymbol{M}$

Case 1: $\boldsymbol{p}$ never proposed to $\boldsymbol{r}$.

$\Rightarrow \boldsymbol{p}$ prefers $\boldsymbol{M}$-partner to $\boldsymbol{r}$.

$\Rightarrow \boldsymbol{p}$-$\boldsymbol{r}$ is not unstable for $\boldsymbol{M}$.

Case 2: $\boldsymbol{p}$ proposed to $\boldsymbol{r}$.

$\Rightarrow \boldsymbol{r}$ rejected $\boldsymbol{p}$ (right away or later when trading up)

$\Rightarrow \boldsymbol{r}$ prefers $\boldsymbol{M}$-partner to $\boldsymbol{p}$.

$\Rightarrow \boldsymbol{p}$-$\boldsymbol{r}$ is not unstable for $\boldsymbol{M}$. ■

Summary

Stable matching problem: Given n people in each of two groups, and their preferences, find a stable matching if one exists.

Stable: No pair of people both prefer to be with each other rather than with their assigned partner

Gale-Shapley algorithm: Guarantees to find a stable matching for *any* problem instance.

Q: How do we implement GS algorithm efficiently?

Q: If there are multiple stable matchings, which one does GS find?

Implementation for Stable Matching Algorithms

- Input size
  - $N = 2n^2$ words
    - $2n$ people each with a preference list of length n
  - $2n^2 \log n$ bits
    - specifying an ordering for each preference list takes $n \log n$ bits

- Brute force algorithm
  - Try all $n!$ possible matchings
  - Do any of them work?

- Gale-Shapley Algorithm
  - n^2 iterations, each costing constant time as follows …

Propose-And-Reject Algorithm

Propose-and-reject algorithm: [Gale-Shapley 1962]

Intuitive method that guarantees to find a stable matching.

- Members of one group ***P*** make *proposals*, the other group ***R*** *receives* proposals

```
Initialize each person to be free
while (some p in P is free) {
    Choose some free p in P
    r = 1st person on p's preference list to whom p has not yet proposed
    if (r is free)
        tentatively match (p,r)    //p and r both engaged, no longer free
    else if (r prefers p to current tentative match p')
        replace (p',r) by (p,r)    //p now engaged, p' now free
    else
        r rejects p
}
```

Efficient Implementation

How do we get the an $O(n^2)$ time implementation?

Input: Representing members of the two groups P and R and their preferences:

- Assume elements of P (proposers) are numbered $1, \dots, n$.
- Assume elements of R (receivers) are numbered $1', \dots, n'$.
- For each proposer, a list/array of the n receivers, ordered by preference.
- For each receiver, a list/array of the n proposers, ordered by preference.

The matching:

- Maintain two arrays **match**$[p]$, and **match'**$[r]$.
  - set entry to **0** if free
  - if p matched to r then **match**$[p]$=r and **match'**$[r]$=p

Making proposals:

- Maintain a list of free proposers, e.g., in a queue.
- Maintain an array **count**$[p]$ that counts the number of proposals already made by proposer p.

Efficient Implementation

Rejecting/accepting proposals:

- Does receiver r prefer proposer p to proposer p'?
  - Using original preference list would be slow
- For each receiver, create ***inverse*** of preference list of proposers.
- Constant time access for each query after $O(n)$ preprocessing per receiver. $O(n^2)$ total preprocessing cost.

r	1st	2nd	3rd	4th	5th	6th	7th	8th
pref	8	3	7	1	4	5	6	2

Proposer **3** or proposer **6** ?

r	1	2	3	4	5	6	7	8
inverse	4th	8th	2nd	5th	6th	7th	3rd	1st

```
for i = 1 to n
   inverse[pref[i]] = i
```

r prefers proposer **3** to **6**

since `inverse`[3] = **2** < **7** =`inverse`[6]

Propose-And-Reject Algorithm

Propose-and-reject algorithm: [Gale-Shapley 1962]

Intuitive method that guarantees to find a stable matching.

- Members of one group ***P*** make *proposals*, the other group ***R*** *receives* proposals

```
Initialize each person to be free.
while (some p in P is free and hasn't proposed to everyone in R) {
    Choose some such free p in P who hasn't proposed to everyone in R
    r = 1st person on p's preference list to whom p has not yet proposed
    if (r is free)
        tentatively match (p,r) //p and r both engaged, no longer free
    else if (r prefers p to current tentative match p')
        replace (p',r) by (p,r) //p now engaged, p' now free
    else
        r rejects p
}
```