CSE 421 Section 1

Stable Matching

Administrivia & Introductions

Your Section TAs

- TA 1
  - Anything you want to say about yourself
- TA 2
  - content

Homework

- Submissions
  - LaTeX (highly encouraged)
    - overleaf.com
    - template and LaTeX guide posted on course website!
  - Word Editor that supports mathematical equations
- All homeworks will be turned in via Gradescope
- Homeworks typically due on Wednesdays at 11:59pm

- Remember, this quarter we have a **LATE PROBLEMS** policy, instead of a late assignments policy
  - You have up to **10 total problem late days**
  - You can use up to **2 late days per problem**; each part of a late day counts as a day

Announcements & Reminders

- Section Materials
  - Handouts will be provided in each section
  - Worksheets and sample solutions will be available on the course calendar later this evening

- HW1
  - Due Wednesday 10/4 @ 11:59pm

Stable Matching

Stable Matching

Given $2n$ people, in two groups, **P** and **R**, of n people, with each person having a preference list for members of the other group, how can we find a stable matching between them?

Perfect Matching:

- Each person **p** in **P** is paired with exactly one person **r** in **R**
- Each person **r** in **R** is paired with exactly one person **p** in **P**

Stability: No ability to exchange partners

Unstable: An unmatched pair **p-r** is unstable if they both prefer each other to current matches

Stable Matching: perfect matching with no unstable pairs

Gale-Shapley Algorithm

Algorithm to find a stable matching:

```
Initially all p in P and r in R are free
while there is a free p
       Let r be highest on p’s list that p has not proposed to
       if r is free
              match (p, r)  “p and r become engaged”
       else //r is not free
              Let p′ be the current match of r
              if r prefers p to p′
                     unmatch (p′, r)
                     match (p, r)
```

Problem 1 – Gale-Shapley

Consider the following stable matching instance:

p_1: r_3, r_1, r_2, r_4
p_2: r_2, r_1, r_4, r_3
p_3: r_2, r_3, r_1, r_4
p_4: r_3, r_4, r_1, r_2

r_1: p_4, p_1, p_3, p_2
r_2: p_1, p_3, p_2, p_4
r_3: p_1, p_3, p_4, p_2
r_4: p_3, p_1, p_2, p_4

a) Run the Gale-Shapley Algorithm on the instance above. When choosing which free p in **P** to propose next, always choose the one with the smallest index (e.g., if p_1 and p_2 are both free, always choose p_1).

Problem 1 – Gale-Shapley

a) Run the Gale-Shapley Algorithm on the instance above. When choosing which free p in **P** to propose next, always choose the one with the smallest index (e.g., if p_1 and p_2 are both free, always choose p_1).

p_1: r_3, r_1, r_2, r_4
p_2: r_2, r_1, r_4, r_3
p_3: r_2, r_3, r_1, r_4
p_4: r_3, r_4, r_1, r_2

r_1: p_4, p_1, p_3, p_2
r_2: p_1, p_3, p_2, p_4
r_3: p_1, p_3, p_4, p_2
r_4: p_3, p_1, p_2, p_4

p_1 chooses r_3

(p_1, r_3)

Problem 1 – Gale-Shapley

a) Run the Gale-Shapley Algorithm on the instance above. When choosing which free p in **P** to propose next, always choose the one with the smallest index (e.g., if p_1 and p_2 are both free, always choose p_1).

p_1: r_3, r_1, r_2, r_4
p_2: r_2, r_1, r_4, r_3
p_3: r_2, r_3, r_1, r_4
p_4: r_3, r_4, r_1, r_2

r_1: p_4, p_1, p_3, p_2
r_2: p_1, p_3, p_2, p_4
r_3: p_1, p_3, p_4, p_2
r_4: p_3, p_1, p_2, p_4

p_1 chooses r_3 — (p_1, r_3)

p_2 chooses r_2 — $(p_1, r_3), (p_2, r_2)$

Problem 1 – Gale-Shapley

a) Run the Gale-Shapley Algorithm on the instance above. When choosing which free p in **P** to propose next, always choose the one with the smallest index (e.g., if p_1 and p_2 are both free, always choose p_1).

p_1: r_3, r_1, r_2, r_4
p_2: r_2, r_1, r_4, r_3
p_3: r_2, r_3, r_1, r_4
p_4: r_3, r_4, r_1, r_2

r_1: p_4, p_1, p_3, p_2
r_2: p_1, p_3, p_2, p_4
r_3: p_1, p_3, p_4, p_2
r_4: p_3, p_1, p_2, p_4

p_1 chooses r_3	(p_1, r_3)
p_2 chooses r_2	(p_1, r_3), (p_2, r_2)
p_3 chooses r_2	(p_1, r_3), **(p_2, r_2)**, **(p_3, r_2)?**

Problem 1 – Gale-Shapley

a) Run the Gale-Shapley Algorithm on the instance above. When choosing which free p in **P** to propose next, always choose the one with the smallest index (e.g., if p_1 and p_2 are both free, always choose p_1).

p_1: r_3, r_1, r_2, r_4
p_2: r_2, r_1, r_4, r_3
p_3: r_2, r_3, r_1, r_4
p_4: r_3, r_4, r_1, r_2

r_1: p_4, p_1, p_3, p_2
r_2: p_1, p_3, p_2, p_4
r_3: p_1, p_3, p_4, p_2
r_4: p_3, p_1, p_2, p_4

p_1 chooses r_3	(p_1, r_3)
p_2 chooses r_2	$(p_1, r_3), (p_2, r_2)$
p_3 chooses r_2	(p_1, r_3), ~~(p_2, r_2)~~, (p_3, r_2)

Problem 1 – Gale-Shapley

a) Run the Gale-Shapley Algorithm on the instance above. When choosing which free p in **P** to propose next, always choose the one with the smallest index (e.g., if p_1 and p_2 are both free, always choose p_1).

p_1: r_3, r_1, r_2, r_4
p_2: r_2, r_1, r_4, r_3
p_3: r_2, r_3, r_1, r_4
p_4: r_3, r_4, r_1, r_2

r_1: p_4, p_1, p_3, p_2
r_2: p_1, p_3, p_2, p_4
r_3: p_1, p_3, p_4, p_2
r_4: p_3, p_1, p_2, p_4

p_1 chooses r_3	(p_1, r_3)
p_2 chooses r_2	$(p_1, r_3), (p_2, r_2)$
p_3 chooses r_2	(p_1, r_3), ~~(p_2, r_2)~~, (p_3, r_2)
p_2 chooses r_1	$(p_1, r_3), (p_2, r_1), (p_3, r_2)$

Problem 1 – Gale-Shapley

a) Run the Gale-Shapley Algorithm on the instance above. When choosing which free p in **P** to propose next, always choose the one with the smallest index (e.g., if p_1 and p_2 are both free, always choose p_1).

p_1: r_3, r_1, r_2, r_4
p_2: r_2, r_1, r_4, r_3
p_3: r_2, r_3, r_1, r_4
p_4: r_3, r_4, r_1, r_2
r_1: p_4, p_1, p_3, p_2
r_2: p_1, p_3, p_2, p_4
r_3: p_1, p_3, p_4, p_2
r_4: p_3, p_1, p_2, p_4

p_1 chooses r_3	(p_1, r_3)
p_2 chooses r_2	$(p_1, r_3), (p_2, r_2)$
p_3 chooses r_2	(p_1, r_3), ~~(p_2, r_2)~~, (p_3, r_2)
p_2 chooses r_1	$(p_1, r_3), (p_2, r_1), (p_3, r_2)$
p_4 chooses r_3	**(p_1, r_3)**, $(p_2, r_1), (p_3, r_2)$, **(p_4, r_3)?**

Problem 1 – Gale-Shapley

a) Run the Gale-Shapley Algorithm on the instance above. When choosing which free p in **P** to propose next, always choose the one with the smallest index (e.g., if p_1 and p_2 are both free, always choose p_1).

p_1: r_3, r_1, r_2, r_4
p_2: r_2, r_1, r_4, r_3
p_3: r_2, r_3, r_1, r_4
p_4: r_3, r_4, r_1, r_2

r_1: p_4, p_1, p_3, p_2
r_2: p_1, p_3, p_2, p_4
r_3: p_1, p_3, p_4, p_2
r_4: p_3, p_1, p_2, p_4

p_1 chooses r_3	(p_1, r_3)	
p_2 chooses r_2	$(p_1, r_3), (p_2, r_2)$	
p_3 chooses r_2	(p_1, r_3), ~~(p_2, r_2)~~, (p_3, r_2)	
p_2 chooses r_1	$(p_1, r_3), (p_2, r_1), (p_3, r_2)$	
p_4 chooses r_3	$(p_1, r_3), (p_2, r_1), (p_3, r_2)$	(p_4, r_3) failed

Problem 1 – Gale-Shapley

a) Run the Gale-Shapley Algorithm on the instance above. When choosing which free p in **P** to propose next, always choose the one with the smallest index (e.g., if p_1 and p_2 are both free, always choose p_1).

p_1: r_3, r_1, r_2, r_4
p_2: r_2, r_1, r_4, r_3
p_3: r_2, r_3, r_1, r_4
p_4: r_3, r_4, r_1, r_2

r_1: p_4, p_1, p_3, p_2
r_2: p_1, p_3, p_2, p_4
r_3: p_1, p_3, p_4, p_2
r_4: p_3, p_1, p_2, p_4

p_1 chooses r_3	(p_1, r_3)
p_2 chooses r_2	$(p_1, r_3), (p_2, r_2)$
p_3 chooses r_2	$(p_1, r_3),$ ~~(p_2, r_2)~~, (p_3, r_2)
p_2 chooses r_1	$(p_1, r_3), (p_2, r_1), (p_3, r_2)$
p_4 chooses r_3	$(p_1, r_3), (p_2, r_1), (p_3, r_2)$ (p_4, r_3) failed
p_4 chooses r_4	$(p_1, r_3), (p_2, r_1), (p_3, r_2), (p_4, r_4)$

Problem 1 – Gale-Shapley

a) Run the Gale-Shapley Algorithm on the instance above. When choosing which free p in **P** to propose next, always choose the one with the smallest index (e.g., if p_1 and p_2 are both free, always choose p_1).

p_1: r_3, r_1, r_2, r_4
p_2: r_2, r_1, r_4, r_3
p_3: r_2, r_3, r_1, r_4
p_4: r_3, r_4, r_1, r_2

r_1: p_4, p_1, p_3, p_2
r_2: p_1, p_3, p_2, p_4
r_3: p_1, p_3, p_4, p_2
r_4: p_3, p_1, p_2, p_4

p_1 chooses r_3	(p_1, r_3)
p_2 chooses r_2	(p_1, r_3), (p_2, r_2)
p_3 chooses r_2	(p_1, r_3), ~~(p_2, r_2)~~, (p_3, r_2)
p_2 chooses r_1	(p_1, r_3), (p_2, r_1), (p_3, r_2)
p_4 chooses r_3	(p_1, r_3), (p_2, r_1), (p_3, r_2) (p_4, r_3) failed
p_4 chooses r_4	(p_1, r_3), (p_2, r_1), (p_3, r_2), (p_4, r_4)

(p_1, r_3), (p_2, r_1), (p_3, r_2), (p_4, r_4)

Problem 1 – Gale-Shapley

b) Run the Gale-Shapley Algorithm on the instance above. When choosing which free p in **P** to propose next, always choose the one with the *largest* index (e.g., if p_1 and p_2 are both free, always choose p_2). Do you get the same result?

p_1: r_3, r_1, r_2, r_4
p_2: r_2, r_1, r_4, r_3
p_3: r_2, r_3, r_1, r_4
p_4: r_3, r_4, r_1, r_2

r_1: p_4, p_1, p_3, p_2
r_2: p_1, p_3, p_2, p_4
r_3: p_1, p_3, p_4, p_2
r_4: p_3, p_1, p_2, p_4

c) Now run the algorithm with the same preferences but with the roles of **P** and **R** reversed (that is the r_i do the proposing) breaking ties by taking the free r_i with the smallest index i. Do you get the same result?

Work on parts b and c of this problem with the people around you, and then we'll go over it together!

Problem 1 – Gale-Shapley

b) Run the Gale-Shapley Algorithm on the instance above. When choosing which free p in **P** to propose next, always choose the one with the *largest* index (e.g., if p_1 and p_2 are both free, always choose p_2). Do you get the same result?

p_1: r_3, r_1, r_2, r_4
p_2: r_2, r_1, r_4, r_3
p_3: r_2, r_3, r_1, r_4
p_4: r_3, r_4, r_1, r_2

r_1: p_4, p_1, p_3, p_2
r_2: p_1, p_3, p_2, p_4
r_3: p_1, p_3, p_4, p_2
r_4: p_3, p_1, p_2, p_4

Problem 1 – Gale-Shapley

b) Run the Gale-Shapley Algorithm on the instance above. When choosing which free p in **P** to propose next, always choose the one with the *largest* index (e.g., if p_1 and p_2 are both free, always choose p_2). Do you get the same result?

p_1: r_3, r_1, r_2, r_4
p_2: r_2, r_1, r_4, r_3
p_3: r_2, r_3, r_1, r_4
p_4: r_3, r_4, r_1, r_2

r_1: p_4, p_1, p_3, p_2
r_2: p_1, p_3, p_2, p_4
r_3: p_1, p_3, p_4, p_2
r_4: p_3, p_1, p_2, p_4

The steps of the Gale-Shapley Algorithm with the free p in **P** with largest index proposing first:

p_4 chooses r_3	(p_4, r_3)	
p_3 chooses r_2	$(p_3, r_2),(p_4, r_3)$	
p_2 chooses r_2	$(p_3, r_2),(p_4, r_3)$	(p_2, r_2) failed
p_2 chooses r_1	$(p_2, r_1),(p_3, r_2),(p_4, r_3)$	
p_1 chooses r_3	$(p_1, r_3),(p_2, r_1),(p_3, r_2)$,~~$(p_4, r_3)$~~	
p_4 chooses r_4	$(p_1, r_3),(p_2, r_1),(p_3, r_2),(p_4, r_4)$	

We ended up with the same result!

Problem 1 – Gale-Shapley

c) Now run the algorithm with the people in **R** proposing, breaking ties by taking the free r_i with the smallest index. Do you get the same result?

$\mathbf{p_1}$: r_3, r_1, r_2, r_4
$\mathbf{p_2}$: r_2, r_1, r_4, r_3
$\mathbf{p_3}$: r_2, r_3, r_1, r_4
$\mathbf{p_4}$: r_3, r_4, r_1, r_2

$\mathbf{r_1}$: p_4, p_1, p_3, p_2
$\mathbf{r_2}$: p_1, p_3, p_2, p_4
$\mathbf{r_3}$: p_1, p_3, p_4, p_2
$\mathbf{r_4}$: p_3, p_1, p_2, p_4

Problem 1 – Gale-Shapley

c) Now run the algorithm with the people in **R** proposing, breaking ties by taking the free r_i with the smallest index. Do you get the same result?

p_1: r_3, r_1, r_2, r_4
p_2: r_2, r_1, r_4, r_3
p_3: r_2, r_3, r_1, r_4
p_4: r_3, r_4, r_1, r_2

r_1: p_4, p_1, p_3, p_2
r_2: p_1, p_3, p_2, p_4
r_3: p_1, p_3, p_4, p_2
r_4: p_3, p_1, p_2, p_4

The steps of the Gale-Shapley Algorithm with the r in **R** proposing:

r_1 chooses p_4	(p_4, r_1)	
r_2 chooses p_1	$(p_1, r_2),(p_4, r_1)$	
r_3 chooses p_1	$(p_1, r_3),$~~(p_1, r_2)~~$,(p_4, r_1)$	
r_2 chooses p_3	$(p_1, r_3),(p_3, r_2),(p_4, r_1)$	
r_4 chooses p_3	$(p_1, r_3),(p_3, r_2),(p_4, r_1)$	(p_3, r_4) failed
r_4 chooses p_1	$(p_1, r_3),(p_3, r_2),(p_4, r_1)$	(p_1, r_4) failed
r_4 chooses p_2	$(p_1, r_3),(p_2, r_4),(p_3, r_2),(p_4, r_1)$	

No, the result is different when we have the r in **R** propose as opposed to the r in **R** .

Induction

Induction

- You will be writing lots of induction proofs in this class in order to prove that your algorithms work the way you say they will.

- The style requirements for proofs in this class are less stringent than the style requirements from 311
  - there is a **style guide** doc on the course website (here) about how 421 proofs are different than what you did in 311

Induction Template

Let $P(n)$ be "(whatever you're trying to prove)".
We show $P(n)$ holds for all n by induction on n.

Base Case: Show $P(b)$ is true.

Inductive Hypothesis: Suppose $P(k)$ holds for an arbitrary $k \geq b$

Inductive Step: Show $P(k+1)$ (i.e. get $P(k) \rightarrow P(k+1)$)

Conclusion: Therefore, $P(n)$ holds for all n by the principle of induction.

Problem 3 – Induction Review

Consider the following claim:

Let $P(n)$ be "Every tree with at least n nodes has at least two nodes of degree-one."

a) What is the correct "skeleton" of the inductive step (i.e., the right things to assume and the right target)?

b) Prove the claim by induction.

Problem 3 – Induction Review

Consider the following claim:

Let $P(n)$ be "Every tree with at least n nodes has at least two nodes of degree-one."

a) What is the correct "skeleton" of the inductive step (i.e., the right things to assume and the right target)?

Work on this problem with the people around you, and then we'll go over it together!

Problem 3 – Induction Review

Consider the following claim:

Let $P(n)$ be "Every tree with at least n nodes has at least two nodes of degree-one."

a) What is the correct "skeleton" of the inductive step (i.e., the right things to assume and the right target)?

Problem 3 – Induction Review

Consider the following claim:

Let $P(n)$ be "Every tree with at least n nodes has at least two nodes of degree-one."

a) What is the correct "skeleton" of the inductive step (i.e., the right things to assume and the right target)?

We must start with "Let T' be an arbitrary tree with $k+1$ nodes."
Our conclusion will be that T' has at least two nodes of degree-one, so $P(k+1)$ holds.

KEY Induction Concept

It might be really tempting to structure the inductive step of this problem as something like, “start with an arbitrary tree T of size k nodes, and then add a node to it, making tree T' with $k + 1$ nodes.”

This is a ***BAD*** idea! Then we’d have to cover every possible way to add on a node (and prove that we had actually dealt with every possible case), making the overall proof way more complicated and unwieldly.

Instead, we ***ALWAYS*** want to start with the bigger thing (in this case, with the arbitrary tree T' of size $k + 1$) and find the smaller thing inside of it.

Problem 3 – Induction Review

Consider the following claim:

Let $P(n)$ be "Every tree with at least n nodes has at least two nodes of degree-one."

b) Prove the claim by induction.

Work on this problem with the people around you, and then we'll go over it together!

Problem 3 – Induction Review

b) Prove the claim by induction.

Let P(n) be "Every tree with at least n nodes has at least two nodes of degree-one." We prove the claim by induction on n.

Base Case: $n = 3$. There is only one undirected tree with three nodes. It has two nodes of degree-one.

Inductive Hypothesis: Suppose $P(n)$ holds for $n = 3, \ldots, k$ for an arbitrary $k \geq 3$.

Problem 3 – Induction Review

b) Prove the claim by induction.

Inductive Step: Let T' be an arbitrary tree with $k+1$ nodes. Let u be a vertex of T' of degree-one (this first vertex exists by the fact), and call its neighbor v. Let T'' be the tree created by deleting u from T'.

Observe that, since u was degree-one, the only simple paths that used (u, v) had u as an endpoint (as once we use (u, v) to arrive at/leave u we cannot reuse it to leave/arrive). Thus T'' is still a connected tree, and we can apply the IH to T'' to conclude there are at least two vertices w_1, w_2 of T'' that are degree-one.

We now find the two degree-one nodes in the original tree T'. We know that u has degree-one (and is not the same as w_1 or w_2 since u was deleted to create T''). Since u has degree-one, it can only attach to at most one of w_1, w_2, thus at least one (the other one) of w_1, w_2 is an additional node of degree-one, as required.

Therefore, T' has the required degree-one vertices. Since T' is an arbitrary tree with $k+1$ vertices, we have shown $P(k+1)$.

Proof or Counterexample?

Prove or Disprove?

Often, you will be given a statement, and then asked to either prove or disprove it. This can be stressful! How do you know which you should start with?

The best way to begin, especially when you don't know if the claim is even true, is to try to understand it better by producing some examples. This has two main benefits that will help, whether you end up proving or disproving the claim:

1) You get a better understanding of the statement so now you have a clear method of approach, OR
2) You find a counterexample, which allows you to easily write a quick proof that the statement is false!

Problem 2 – A Quick Proof

Is it possible to have a stable matching instance with more than 2 stable matchings? If so, give an instance and at least 3 stable matchings. If not, prove that every instance has at most 2 stable matchings.

Work on this problem with the people around you, and then we'll go over it together!

Problem 2 – A Quick Proof

Is it possible to have a stable matching instance with more than 2 stable matchings? If so, give an instance and at least 3 stable matchings. If not, prove that every instance has at most 2 stable matchings.

Problem 2 – A Quick Proof

Is it possible to have a stable matching instance with more than 2 stable matchings? If so, give an instance and at least 3 stable matchings. If not, prove that every instance has at most 2 stable matchings.

Consider the following instance:

$\mathbf{p_1}$: r_1, r_2, r_3, r_4
$\mathbf{p_2}$: r_2, r_1, r_4, r_3
$\mathbf{p_3}$: r_3, r_4, r_1, r_2
$\mathbf{p_4}$: r_4, r_3, r_2, r_1

$\mathbf{r_1}$: p_2, p_1, p_4, p_3
$\mathbf{r_2}$: p_1, p_2, p_3, p_4
$\mathbf{r_3}$: p_4, p_3, p_2, p_1
$\mathbf{r_4}$: p_3, p_4, p_1, p_2

This instance has four stable matchings:

$(p_1, r_1),(p_2, r_2),(p_3, r_3),(p_4, r_4)$
$(p_1, r_1),(p_2, r_2),(p_3, r_4),(p_4, r_3)$
$(p_1, r_2),(p_2, r_1),(p_3, r_3),(p_4, r_4)$
$(p_1, r_2),(p_2, r_1),(p_3, r_4),(p_4, r_3)$

That’s All, Folks!

Thanks for coming to section this week!
Any questions?