CSE 421 Section 1

Stable Matching

Administrivia & Introductions

Your TAs

- Allie Pfleger
  - BS/MS student, transferred in Fall of 2020
  - 8th time TA, first time 421!!
  - I'm married and I have a 5-year-old daughter Sophia
  - I love knitting, reading, playing games, and watching fun movies and shows!
- Airei Fukuzawa (eye-ray)
  - 3rd year undergrad
  - 1st time TA (Please tolerate me ☺)
  - Took 421 Spring22
  - I love eating, traveling, and live music!

Homework

- Submissions
  - LaTeX (highly encouraged)
    - overleaf.com
    - template and LaTeX guide posted on course website!
  - Word Editor that supports mathematical equations
  - Handwritten neatly and scanned
- All homeworks will be turned in via Gradescope
- Homeworks typically due on Wednesdays at 10pm

Announcements & Reminders

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

- HW1
  - Due Wednesday 10/5 @ 10pm

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

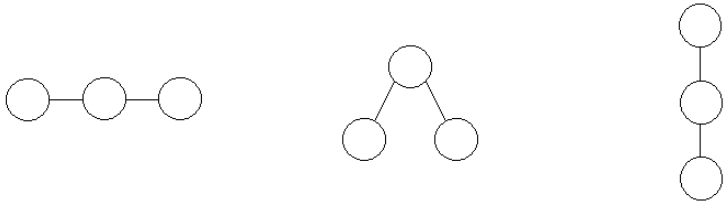
Problem 2 - Find The Bug: Spoof By Induction

What follows is an **incorrect** proof by induction.

Claim: Every (undirected) tree with at least three nodes has at least two nodes of degree-one.

Spoof. Let P(n) be “Every tree with at least n nodes has at least two nodes of degree-one.”

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 2 - Find The Bug: Spoof By Induction

Inductive Step: let T be an arbitrary tree with k nodes. By inductive hypothesis, T has at least two nodes of degree-one. Call them u and v.

We now build T', which has $k+1$ nodes. Take T, and create a new node w. Since we are interested in connected trees, we must connect w; we break into cases depending on what it is adjacent to.

Case 1: w is adjacent to neither u nor v. If w is adjacent to a node other than u, v then u and v still have degree-one, so the claim holds on T'.

Case 2: w is adjacent to one of u, v but not the other. If w is adjacent to u or v, then the other of u, v and w will both be degree-one

Case 3: w is adjacent to both u, v. This case is impossible! If w were connected to both u and v, then the path in T between u and v (which exists because T was connected) along with (u, w) and (v, w) form a cycle, which is not allowed in a tree.

In all (allowed) cases, T' has the required degree-one vertices. Since we constructed T' to have $k+1$ vertices, we have shown $P(k+1)$.

Problem 2 - Find The Bug: Spoof By Induction

a) Clearly describe the bug and why the proof is incorrect.

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

c) Is the claim true? If so, write a correct proof. If not, provide a counter-example. You may use the fact “every tree has at least one node of degree-one” (this fact is just the Claim from Problem ?? in contrapositive form).

Work on (a) and (b) of this problem with the people around you, and then we’ll go over it together!

Problem 2 - Find The Bug: Spoof By Induction

a) Clearly describe the bug and why the proof is incorrect.

Problem 2 - Find The Bug: Spoof By Induction

a) Clearly describe the bug and why the proof is incorrect.

We never introduce an arbitrary tree with $k + 1$ nodes in the inductive step! Why should our reader believe we have handled all possibilities between our three cases? Notice that we don't have a recursive definition of "tree" – we aren't doing structural induction here! We need to start with an arbitrary tree with $k + 1$ nodes. That's how you show a statement of the form "for all trees with $k + 1$ nodes..."

Problem 2 - Find The Bug: Spoof By Induction

a) Clearly describe the bug and why the proof is incorrect.

We never introduce an arbitrary tree with $k + 1$ nodes in the inductive step! Why should our reader believe we have handled all possibilities between our three cases? Notice that we don't have a recursive definition of "tree" – we aren't doing structural induction here! We need to start with an arbitrary tree with $k + 1$ nodes. That's how you show a statement of the form "for all trees with $k + 1$ nodes..."

It turns out that we really have addressed all cases here – every tree really can be built with these rules – but without a recursive definition, we'd need to write a detailed proof to explain that every tree really can be built that way first. It's not worth it.

Problem 2 - Find The Bug: Spoof By Induction

a) Clearly describe the bug and why the proof is incorrect.

We never introduce an arbitrary tree with $k + 1$ nodes in the inductive step! Why should our reader believe we have handled all possibilities between our three cases? Notice that we don't have a recursive definition of "tree" – we aren't doing structural induction here! We need to start with an arbitrary tree with $k + 1$ nodes. That's how you show a statement of the form "for all trees with $k + 1$ nodes..."

It turns out that we really have addressed all cases here – every tree really can be built with these rules – but without a recursive definition, we'd need to write a detailed proof to explain that every tree really can be built that way first. It's not worth it.

When proving a for-all statement by induction always start with the big thing and find the smaller thing inside.

Problem 2 - Find The Bug: Spoof By Induction

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

Problem 2 - Find The Bug: Spoof By Induction

b) 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.”

Problem 2 - Find The Bug: Spoof By Induction

b) 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.

Problem 2 - Find The Bug: Spoof By Induction

c) Is the claim true? If so, write a correct proof. If not, provide a counter-example. You may use the fact “every tree has at least one node of degree-one” (this fact is just the Claim from Problem ?? in contrapositive form).

Problem 2 - Find The Bug: Spoof By Induction

c) Is the claim true? If so, write a correct proof. If not, provide a counter-example. You may use the fact “every tree has at least one node of degree-one” (this fact is just the Claim from Problem ?? in contrapositive form).

Proof. The proof is identical except for the IS:

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 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.

Proof by Contradiction

Contradiction

- In addition to induction, proof by contradiction is another really common technique we will use to prove that algorithms are correct in this class.

- Explicitly state that you are doing proof by contradiction in the introduction of your proof!

- Explicitly identify what it is that you are supposing!

Problem 1 - Write it Better: Proof by Contradiction

What follows is a **correct**, but *poorly presented*, proof by contradiction. We'll show two different ways to make it cleaner.

Claim: For every simple graph G, if every node of G has degree at least 2, then G has a cycle.

An unclear proof. Suppose, for the sake of contradiction, there is a graph G such that every node of G has degree at least 2, but G has no cycle.

We will construct a simple path in G.

Start at some node v_0, of G. Follow v_0 along an edge to find v_1. Now, since v_1 (like every other node) has degree at least 2, there is another edge attached to v_1. Follow it to a vertex v_2. If v_2 is a vertex we have already visited (i.e., v_0), then we have found a cycle, a contradiction! Otherwise, from v_2, we may repeat the same argument. Continue from v_2 (also degree at least 2) to a vertex v_3. If v_3 is a repeat (v_0 or v_1) then we have a contradiction! Otherwise continue finding v_4, v_5, ... The graph is finite, so we cannot continue this process forever. Eventually we find a repeated vertex, which means we have a cycle, a contradiction!

Problem 1 - Write it Better: Proof by Contradiction

a) It's common in proofs by contradiction to have cases like we've seen here; "*Option A*: we're done with our proof! *Option B*: do something else." Here, though, that "do something else" has us basically where we started (a new end-vertex on our path where we've used one edge), and it's tempting to say "repeat indefinitely, eventually you hit the other case." That's mathematically correct! But not particularly elegant. And you have to write down enough steps that your reader knows what the pattern is, which could be a lot.

The more elegant version is to use ***proof by contradiction with extremality***. Instead of slowly building an object (here, the path), just start with the most *extreme* version of the object at the beginning (usually the biggest one or the first one). Starting with the right object lets us eliminate Option A and jump right to Option B.

Let's see if this proof is any cleaner. Finish the proof:

Proof. Suppose, for the sake of contradiction, there is a graph G such that every node of G has degree at least 2, but G has no cycle. Let $P = v_0, v_1, \dots, v_k$ be a longest simple path in G.

Problem 1 - Write it Better: Proof by Contradiction

a) Finish the proof:

Proof. Suppose, for the sake of contradiction, there is a graph G such that every node of G has degree at least 2, but G has no cycle. Let $P = v_0, v_1, \ldots, v_k$ be a longest simple path in G.

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

Problem 1 - Write it Better: Proof by Contradiction

a) Finish the proof:

Proof. Suppose, for the sake of contradiction, there is a graph G such that every node of G has degree at least 2, but G has no cycle. Let $P = v_0, v_1, \ldots, v_k$ be a longest simple path in G.

Since v_0 has degree at-least 2, it must have another neighbor, w other than v_1.

Since P is a longest simple path, w must be a repeat of a vertex among $v_1, \ldots, v_k$ (otherwise $w, v_0, v_1, \ldots v_k$ would be a longer simple path).

Combining the edge (v_0, w) with P from v_0 to w gives a cycle.
But G was acyclic, that's a contradiction!

Gale-Shapley Algorithm

Stable Matching

Given n riders and n horses with preference lists, how can we find a stable matching so all riders have horses and all horses have riders?

Perfect Matching:

- Each rider is paired with exactly one horse
- Each horse is paired with exactly one rider

Stability: No ability to exchange partners

Blocking: An unmatched pair r-h is blocking if they both prefer each other to current matches

Stable Matching: perfect matching with no blocking pairs

Gale-Shapley Algorithm

Algorithm to find a stable matching: (we will prove it works in lecture)

```
Initially all r in R and h in H are free
while there is a free r
        Let h be highest on r's list that r has not proposed to
        if h is free
                match (r,h)
        else //h is not free
                Let r' be the current match of h
                if h prefers r to r'
                        unmatch (r',h)
                        match (r,h)
```

Problem 5 – Gale-Shapley

Consider the following stable matching instance:

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

Problem 5 – Gale-Shapley

a) Run the Gale-Shapley Algorithm with riders proposing on the instance above. When choosing which free rider to propose next, always choose the one with the smallest index (e.g., if r_1 and r_2 are both free, always choose r_1).

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

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

Problem 5 – Gale-Shapley

a) Run the Gale-Shapley Algorithm with riders proposing on the instance above. When choosing which free rider to propose next, always choose the one with the smallest index (e.g., if r_1 and r_2 are both free, always choose r_1).

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

r_1 chooses h_3 (r_1, h_3)

Problem 5 – Gale-Shapley

a) Run the Gale-Shapley Algorithm with riders proposing on the instance above. When choosing which free rider to propose next, always choose the one with the smallest index (e.g., if r_1 and r_2 are both free, always choose r_1).

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

r_1 chooses h_3 — (r_1, h_3)

r_2 chooses h_2 — $(r_1, h_3), (r_2, h_2)$

Problem 5 – Gale-Shapley

a) Run the Gale-Shapley Algorithm with riders proposing on the instance above. When choosing which free rider to propose next, always choose the one with the smallest index (e.g., if r_1 and r_2 are both free, always choose r_1).

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

r_1 chooses h_3	(r_1, h_3)
r_2 chooses h_2	$(r_1, h_3), (r_2, h_2)$
r_3 chooses h_2	$(r_1, h_3), \mathbf{(r_2, h_2)}, \mathbf{(r_3, h_2)}$

Problem 5 – Gale-Shapley

a) Run the Gale-Shapley Algorithm with riders proposing on the instance above. When choosing which free rider to propose next, always choose the one with the smallest index (e.g., if r_1 and r_2 are both free, always choose r_1).

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

r_1 chooses h_3	(r_1, h_3)
r_2 chooses h_2	$(r_1, h_3), (r_2, h_2)$
r_3 chooses h_2	$(r_1, h_3), \cancel{(r_2, h_2)}, (r_3, h_2)$

Problem 5 – Gale-Shapley

a) Run the Gale-Shapley Algorithm with riders proposing on the instance above. When choosing which free rider to propose next, always choose the one with the smallest index (e.g., if r_1 and r_2 are both free, always choose r_1).

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

r_1 chooses h_3	(r_1, h_3)
r_2 chooses h_2	$(r_1, h_3), (r_2, h_2)$
r_3 chooses h_2	$(r_1, h_3), \cancel{(r_2, h_2)}, (r_3, h_2)$
r_2 chooses h_1	$(r_1, h_3), (r_2, h_1), (r_3, h_2)$

Problem 5 – Gale-Shapley

a) Run the Gale-Shapley Algorithm with riders proposing on the instance above. When choosing which free rider to propose next, always choose the one with the smallest index (e.g., if r_1 and r_2 are both free, always choose r_1).

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

r_1 chooses h_3	(r_1, h_3)
r_2 chooses h_2	$(r_1, h_3), (r_2, h_2)$
r_3 chooses h_2	$(r_1, h_3), \cancel{(r_2, h_2)}, (r_3, h_2)$
r_2 chooses h_1	$(r_1, h_3), (r_2, h_1), (r_3, h_2)$
r_4 chooses h_3	$\mathbf{(r_1, h_3)}, (r_2, h_1), (r_3, h_2), \mathbf{(r_4, h_3)}$

Problem 5 – Gale-Shapley

a) Run the Gale-Shapley Algorithm with riders proposing on the instance above. When choosing which free rider to propose next, always choose the one with the smallest index (e.g., if r_1 and r_2 are both free, always choose r_1).

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

r_1 chooses h_3	(r_1, h_3)
r_2 chooses h_2	$(r_1, h_3), (r_2, h_2)$
r_3 chooses h_2	$(r_1, h_3), \cancel{(r_2, h_2)}, (r_3, h_2)$
r_2 chooses h_1	$(r_1, h_3), (r_2, h_1), (r_3, h_2)$
r_4 chooses h_3	$(r_1, h_3), (r_2, h_1), (r_3, h_2), \cancel{(r_4, h_3)}$

Problem 5 – Gale-Shapley

a) Run the Gale-Shapley Algorithm with riders proposing on the instance above. When choosing which free rider to propose next, always choose the one with the smallest index (e.g., if r_1 and r_2 are both free, always choose r_1).

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

r_1 chooses h_3	(r_1, h_3)
r_2 chooses h_2	$(r_1, h_3), (r_2, h_2)$
r_3 chooses h_2	$(r_1, h_3), \cancel{(r_2, h_2)}, (r_3, h_2)$
r_2 chooses h_1	$(r_1, h_3), (r_2, h_1), (r_3, h_2)$
r_4 chooses h_3	$(r_1, h_3), (r_2, h_1), (r_3, h_2), \cancel{(r_4, h_3)}$
r_4 chooses h_4	$(r_1, h_3), (r_2, h_1), (r_3, h_2), (r_4, h_4)$

Problem 5 – Gale-Shapley

a) Run the Gale-Shapley Algorithm with riders proposing on the instance above. When choosing which free rider to propose next, always choose the one with the smallest index (e.g., if r_1 and r_2 are both free, always choose r_1).

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

r_1 chooses h_3	(r_1, h_3)
r_2 chooses h_2	$(r_1, h_3), (r_2, h_2)$
r_3 chooses h_2	$(r_1, h_3), \cancel{(r_2, h_2)}, (r_3, h_2)$
r_2 chooses h_1	$(r_1, h_3), (r_2, h_1), (r_3, h_2)$
r_4 chooses h_3	$(r_1, h_3), (r_2, h_1), (r_3, h_2), \cancel{(r_4, h_3)}$
r_4 chooses h_4	$(r_1, h_3), (r_2, h_1), (r_3, h_2), (r_4, h_4)$

$(r_1, h_3), (r_2, h_1), (r_3, h_2), (r_4, h_4)$

That's All, Folks!

Thanks for coming to section this week!
Any questions?