

CSCI 256 Lecture 4:

Stable Matching & Asymptotics

Announcements

- Hand in Exercise #3, Pick up Exercise #4
- Need to use LaTeX **and** Overleaf template provided!
 - No handwritten
 - Need to follow guidelines for full credit
- Assignment 1 released and due Wed (9/23) at 10 pm
 - Work in pairs: invite your partner to your Overleaf project
- Joint write up submitted to Gradescope

Group Members

[?](#) Add or remove group members for this submission.

Your instructor has allowed you to submit as a group of up to any number of people. You can change the group below. Students added or removed will be notified via email.

Student	Remove
Katy Student (Submitter)	
Byron Washington	

Add Student

Last Time & Today

- Defined asymptotic notation
 - Big O : upper bound (analog of $\leq$)
 - Big Ω : lower bound (analog of $\geq$)
 - Big Θ : tight equality up to lower-order/constants (analog of $=$)
- Today:
 - Apply Big O and analyze Gale Shapley running time
 - Tips on how to compare functions using these notations

Gale-Shapely Algorithm

GALE-SHAPLEY (*preference lists for hospitals and students*)

INITIALIZE M to empty matching.

WHILE (some hospital h is unmatched and hasn't proposed to every student)

$s \leftarrow$ first student on h 's list to whom h has not yet proposed.

IF (s is unmatched)

 Add $h-s$ to matching M .

ELSE IF (s prefers h to current partner h')

 Replace $h'-s$ with $h-s$ in matching M .

ELSE

s rejects h .

RETURN stable matching M .

Gale-Shapely Algorithm

GALE-SHAPLEY (*preference lists for hospitals and students*)

INITIALIZE M to empty matching.

WHILE (some hospital h is unmatched and hasn't proposed to every student)

$s \leftarrow$ first student on h 's list to whom h has not yet proposed.

IF (s is unmatched)

 Add $h-s$ to matching M .

ELSE IF (s prefers h to current partner h')

 Replace $h'-s$ with $h-s$ in matching M .

ELSE

s rejects h .

RETURN stable matching M .

Number of iterations?

How long does each iteration take?

Analyzing Time Complexity

- Each hospital makes an offer to each student at most once
- Upper bound on total number of offers that can ever be made?
 - At most n^2 iterations
- What do we do in each iteration?

Gale-Shapely Algorithm

GALE-SHAPLEY (*preference lists for hospitals and students*)

INITIALIZE M to empty matching.

WHILE (some hospital h is unmatched and hasn't proposed to every student)

$s \leftarrow$ first student on h 's list to whom h has not yet proposed.

IF (s is unmatched)

Add $h-s$ to matching M .

ELSE IF (s prefers h to current partner h')

Replace $h'-s$ with $h-s$ in matching M .

ELSE

s rejects h .

RETURN stable matching M .

Find an unmatched h

Find top student in h 's list

Check matched status,
update matching

check if s prefers h to h'

Analyzing Time Complexity

- What do we do in each iteration?
 - Select a free hospital h
 - Find top ranked s not yet offered a post by h
 - Check if s is matched
 - Compare s 's ranking of h with its current match
 - Add to & delete from set of matched pairs
- Let's analyze how long each of these steps take

Analyzing Time Complexity

- How long does it take to find an unmatched hospital?
- Can we use a data structure to do this in $O(1)$ time?
 - Store unmatched hospitals in a queue (implemented using a doubly linked list)
 - Removing an unmatched hospital from queue and adding new unmatched hospital to it are both $O(1)$
- How long to find the next student h must propose to?
 - Can store in a linked list and pop/remove first each time
 - $O(1)$ time operation

Gale-Shapely Algorithm

GALE-SHAPLEY (*preference lists for hospitals and students*)

INITIALIZE M to empty matching.

WHILE (some hospital h is unmatched and hasn't proposed to every student)

$s \leftarrow$ first student on h 's list to whom h has not yet proposed.

IF (s is unmatched)

 Add $h-s$ to matching M .

ELSE IF (s prefers h to current partner h')

 Replace $h'-s$ with $h-s$ in matching M .

ELSE

s rejects h .

RETURN stable matching M .

Find an unmatched h

Find top student in h 's list

Check matched status,
update matching

check if s prefers h to h'

Analyzing Time Complexity

- How to maintain current matching and check for match status?
- Index an array using student indices $1, \dots, n$
- Add to & delete from set of matched pairs:
 - Array, $\text{Matched}(s) = h$ currently matched to s (or 'free')
- How much time to create?
- How much time to update?

Gale-Shapely Algorithm

GALE-SHAPLEY (*preference lists for hospitals and students*)

INITIALIZE M to empty matching.

WHILE (some hospital h is unmatched and hasn't proposed to every student)

$s \leftarrow$ first student on h 's list to whom h has not yet proposed.

IF (s is unmatched)

 Add $h-s$ to matching M .

ELSE IF (s prefers h to current partner h')

 Replace $h'-s$ with $h-s$ in matching M .

ELSE

s rejects h .

RETURN stable matching M .

Find an unmatched h

Find top student in h 's list

Check matched status,
update matching

check if s prefers h to h'

Analyzing Time Complexity

- How to check if s prefer h to the current hospital h' ?
 - If we scan its preference list, it will take $O(n)$ time
 - Can we do this more efficiently?
- "Preprocess" student preference lists to make this faster
- For each s , create **inverse of preference list** of hospitals

student prefers hospital 4 to 6 since $\text{rank}[4] < \text{rank}[6]$

Student preference list indexed by rank

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

hospital indices

Inverse pref-list indexed by hospital

	1	2	3	4	5	6	7	8
rank[]	4 th	8 th	2 nd	5 th	6 th	7 th	3 rd	1 st

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

Analyzing Time Complexity

- For each s , create **inverse of preference list** of hospitals
- Once we have inverse list, this step takes $O(1)$ time to compare to hospitals
- How long does this preprocessing step take?
 - $O(n)$ time for each student, $O(n^2)$ overall

student prefers hospital 4 to 6 since $\text{rank}[4] < \text{rank}[6]$

Student preference list indexed by rank

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

Inverse pref-list indexed by hospital

	1	2	3	4	5	6	7	8
rank[]	4 th	8 th	2 nd	5 th	6 th	7 th	3 rd	1 st

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

Gale-Shapely Algorithm

GALE-SHAPLEY (*preference lists for hospitals and students*)

$O(1)$

Find an unmatched h

INITIALIZE M to empty matching.

WHILE (some hospital h is unmatched and hasn't proposed to every student)

$s \leftarrow$ first student on h 's list to whom h has not yet proposed.

$O(1)$

IF (s is unmatched)

Add $h-s$ to matching M .

Find top student in h 's list

ELSE IF (s prefers h to current partner h')

Replace $h'-s$ with $h-s$ in matching M .

$O(1)$

Check matched status,
update matching

ELSE

s rejects h .

check if s prefers h to h'

RETURN stable matching M .

$O(1)$

$O(n^2)$

Analyzing Time Complexity

Analyzing overall running time:

- Preprocessing to create inverse list: $O(n^2)$
- Maintain free hospitals: Queue: $O(1)$ for `get()` and `put()`
- Match and check match status: $O(1)$
- Each iteration thus takes $O(1)$ time
- Overall, $O(n^2)$ time preprocessing + $O(n^2)$ time in iterations
- **Question.** Why is this the best we can hope for?
 - Input size is $O(n^2)$ size, *linear in input size*

Comparing Asymptotic Complexity

Simplifying Functions First

- Compare $f(n) = 100n^2 + n + 10$ and $g(n) = n^3$
- Simplify $f(n)$ as much as possible by keeping highest order term

$$100n^2 + n + 100 < 100n^2 + n + n \quad \text{for any } n > 100$$

$$< 100n^2 + 2n^2 < 102n^2 \quad \text{for any } n > 100$$

- Now easier to compare $102n^2$ with n^3
 - $102n^2 < n^3$ for any $n > 102$

Definition (Alternate). $f(n)$ is $O(g(n))$ if there exists constant $C > 0$ s.t.

$$0 \leq f(n) \leq C \cdot g(n) \quad \text{for all } n \geq C$$

Tools for Comparing Asymptotics

- **Transitivity:** If $f(n) = O(g(n))$ and $g(n) = O(h(n))$ then

$$f(n) = O(h(n))$$

- **Proof.** There exists $C_1, C_2 > 0$ such that

$$f(n) \leq C_1 g(n) \quad \text{for } n > C_1 \quad \text{and} \quad g(n) \leq C_2 h(n) \quad \text{for } n > C_2$$

- Thus, $f(n) \leq C_{\max} h(n)$ for $n > C_{\max}$ where
 $C_{\max} = \max\{C_1, C_2\}$

Tools for Comparing Asymptotics

- Logs and polynomial of logs grow slowly than any polynomial:
- Polynomials grow slowly than any exponent
- Base matters in exponents
- Can you limits to decide:

You can apply these without proof

- If $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$, then $f(n) = O(g(n))$

- If $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c$ for some constant $0 < c < \infty$, then

$$f(n) = \Theta(g(n))$$

Tools for Comparing Asymptotics

- Polynomials grow slowly than any exponent:
 - $n^a = O(b^n)$ for every $a \geq 0$, $b > 1$
- Example
 - $n^3 = O(3^n)$
 - $n^{1/10} = O(1.5^n)$

Tools for Comparing Asymptotics

- Logs and polynomial of logs grow slower than any polynomial:
 - $(\log_b n)^a = O(n^c)$ for every $a, c > 0, b > 1$
- Examples
 - $\sqrt{\log n} = O(n^{0.1})$
 - $\log_{1000} n = O(\sqrt{n})$
 - $(\log n)^{100} = O(n)$

Tools for Comparing Asymptotics

- Exponent base matters:
 - $a^n = O(b^n)$ but $b^n \neq O(a^n)$ for every $b > a > 1$
- Examples
 - 2^n grows strictly slower than 6^n

Simplify Functions First

- Compare $3^{\log n}$ vs 2^n
 - Properties of logs/exponents can help

Simplify Functions First

- Compare $3^{\log n}$ vs 2^n
 - Properties of logs/exponents can help
- Write $3^{\log n} = (2^{\log_2 3})^{\log_2 n}$
$$= (2^{\log_2 n})^{\log_2 3}$$
$$= n^{\log_2 3}$$
$$= n^c$$
 - Now easy to compare n^c to 2^n

Playing with Logs: Properties

- In this class, $\log n$ means $\log_2 n$, $\ln n = \log_e n$
- Constant base doesn't matter: $\log_b(n) = \frac{\log n}{\log b} = \Theta(\log n)$
- Properties of logs:
 - $\log(n^m) = m \log n$
 - $\log(ab) = \log a + \log b$
 - $\log(a/b) = \log a - \log b$

$$a^{\log_a n} = n$$

We will use this a lot!

Exponents

$$n^a \cdot n^b = n^{a+b}$$

$$(n^a)^b = n^{ab}$$

Class Exercise

- Answer True/False by simplifying the function
 - Is $n^{1/\log n} = O(1)$?
 - Is $\log \sqrt{4^n} = O(n^2)$?
 - Is $n = O(2^{\log_4 n})$?

Class Exercise

- When comparing two functions, helpful to simplify first
- Is $n^{1/\log n} = O(1)$?
 - Simplify $n^{1/\log n} = (2^{\log n})^{1/\log n} = 2$: **True**
- Is $\log \sqrt{4^n} = O(n^2)$?
 - Simplify $\log \sqrt{2^{2n}} = \log 2^n = n \log 2 = O(n)$: **True**
- Is $n = O(2^{\log_4 n})$?
 - Simplify $2^{\log_4 n} = 2^{\frac{\log_2 n}{\log_2 4}} = 2^{(\log_2 n)/2} = 2^{\log_2 \sqrt{n}} = \sqrt{n}$:
False